Early Dependability Analysis of FPGA-Based Space Applications Using Formal Verification

Khaza Anuarul Hoque

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

February 2016

© Khaza Anuarul Hoque, 2016

CONCORDIA UNIVERSITY

Division of Graduate Studies

This is to certify that the thesis prepared

By: **Khaza Anuarul Hoque**

Entitled: **Early Dependability Analysis of FPGA-Based Space Applications Using Formal Verification**

and submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________ Dr. Ben Hamza

______________________ Dr. Raoul Velazco

______________________ Dr. Rachida Dssouli

______________________ Dr. Wahab Hamou-Lhadj

______________________ Dr. Sofiène Tahar

______________________ Dr. Otmane Ait Mohamed

______________________ Dr. Yvon Savaria

Approved by ______________________________

Chair of the ECE Department

__________ 2016 ______________________________

Dean of Engineering

ABSTRACT

Early Dependability Analysis of FPGA-Based Space Applications Using Formal Verification

Khaza Anuarul Hoque

Concordia University, 2016

SRAM-based FPGAs are increasingly attractive in the aerospace industry for their field programmability and low cost. Unfortunately, they suffer from cosmic radiation induced Single Event Effects (SEEs). In safety-critical applications, the dependability of the design is a prime concern since failures may have catastrophic consequences. Hence, an early analysis of dependability of such safety-critical applications will enable designers to develop systems that meets high dependability requirements, such as the DO-254 standard. In this thesis, we propose a high-level dependability and performability analysis methodology based on probabilistic model checking. Compared to the pen-and-pencil and discrete-event simulation approach, our methodology is more accurate due to the use of an automated formal verification technique. Moreover, compared to fault injection or beam testing, analysis at early design stages can guide designers to build more reliable designs reducing the overall cost and effort. The proposed methodology can perform three different types of analysis: evaluation of available design options, optimization of scrub intervals while satisfying its design assurance level requirements, and optimal partitioning of Triple-Modular Redundant (TMR) Systems. Such analysis can also guide designers to adopt proper mitigation technique(s), such as rescheduling, TMR, TMR with less frequent scrubs, or even can help to decide the number of TMR partitions for a given scrub intervals.

Starting from a high-level description of a system, based on the preferred analysis, a Markov model or Markov (reward) model is constructed from the extracted Control Data Flow Graph (CDFG) and the failure/mitigation parameters for the targeted FPGA. Such modeling and exhaustive analysis elaborated using a probabilistic model checking technique can capture all the failures and repairs possible (according to some general model) in the system within the radiation environment. To illustrate the applicability of the proposed approach, we present our quantitative analysis obtained from DSP benchmark circuits.

To My Mother, and The Rest of My Family.

ACKNOWLEDGEMENTS

First, I am deeply grateful to Dr. Otmane Ait Mohamed and Dr. Yvon Savaria for their guidance, support and encouragements throughout my graduate studies. Dr. Ait Mohamed taught me the insights of formal verification, and Dr. Savaria has shown me how to apply my formal verification skill in real life problems. I am thankful for those long meetings that we used to have about the correctness of the approach, correctness of the obtained results and how to enhance the work. Working with them has given me the experience and confidence that I can use throughout my future research.

I would like to express my gratitude to Dr. Raoul Velazco (TIMA laboratory, France) for taking time out of his busy schedule to serve as my external examiner. I must also thank the members of the thesis committee for their assistance at all levels of this research project. Their valuable feedback and comments at various stages have been extremely useful in shaping the thesis to completion. This research work is a part of the AVIO-403 project and was financially supported by Consortium for Research and Innovation in Aerospace in Quebec (CRIAQ), Fonds de Recherche du Québec - Nature et Technologies (FRQNT), and Concordia University. Also, many thanks to the industrial collaborators: Bombardier Aerospace, MDA Space Missions and Canadian Space Agency (CSA) for their technical guidance and financial support.

My sincere thanks to all of my family members for being patient and for putting their trust on me. Honestly, they have been an endless source of love, affection, support, and motivation for me. I deeply thank my mother Shamsun Ara Sultana who always encouraged me to pursue my Ph.D. degree. I would also like to thank the members of

the Hardware Verification Group (HVG) for their support and friendliness. Finally, my greatest regards to the Almighty for bestowing upon me the courage to face the complexities of a graduate student's life and to complete this project successfully.

TABLE OF CONTENTS

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ACRONYMS xv

1 Introduction **1**

1.1 Motivation 1

1.2 State-of-the-Art in SEU Analysis 8

1.2.1 Modeling and Analysis of Fault Mitigation Techniques 8

1.2.2 TMR and Scrub Modeling 10

1.2.3 TMR partitioning 11

1.3 Proposed Methodology 12

1.4 Thesis Contributions 15

1.5 Thesis Organization 17

2 Preliminaries **18**

2.1 Single Event Effects 18

2.1.1 Soft Errors 19

2.1.2 Hard Errors 20

2.2 Single Event Upsets in SRAM-based FPGAs 21

2.3 Scrubbing 23

2.4 Dependability Metrics 25

2.4.1 Reliability 25

2.4.2 Availability 26

2.4.3 Safety 26

2.4.4 Performance 27

2.4.5 Performability 27

2.5 Probabilistic Model Checking and PRISM 28

3 Design Option Analysis **32**

3.1 CDFG and High-level Synthesis 33

3.2 CDFG Rescheduling for Fault Recovery 35

3.3 Methodology for design option analysis 36

3.3.1 Markov Modeling of Reliability and Availability 38

3.3.2 Safety Modeling using Fault Coverage 42

3.3.3 Peformability Modeling using MRM 44

Markov Reward Modeling for the CDFG 46

3.3.4 Characterization Library 47

3.4 Quantitative Analysis using PRISM 49

3.5 Summary 60

4 Scrub Optimization and DAL Analysis **62**

4.1 Erlang Distribution 64

4.2 Scrub optimization/DAL Analysis Methodology 66

4.2.1 Resource Estimation 66

4.2.2 Markov Modeling of Failure and Deterministic Delay 67

4.2.3 Modeling Scrub and TMR 70

4.2.4 Markov Model Parameters 72

Environmental parameters 72

Target system parameters 73

Mitigation parameters 74

4.3 Case Study 74
4.3.1 Analysis 76
4.3.2 Verification 79
4.4 Summary 81

5 Optimal Partitioning of TMR 83

5.1 TMR Partitioning 84
5.2 Methodology for TMR partitioning analysis 85
5.2.1 Markov Modeling of Single Bit Upsets (SBUs) 87
5.2.2 Markov Modeling of Multiple Bit Upsets (MBUs) 93
5.3 Quantitative Analysis of an FIR Filter using PRISM 95
5.4 Summary 100

6 Conclusion and Future Work 101

6.1 Conclusions 101
6.2 Future Work 103

Bibliography 106

Biography 120

LIST OF TABLES

1.1 Comparison of FPGA Technologies [20] 3

3.1 Characterization library 48

3.2 Available design options to evaluate 50

3.3 Configurations vs classes of states 51

3.4 Overall reward calculation 58

4.1 Model construction time and statistics 74

4.2 Scrub intervals vs reliability and availability 77

4.3 Verification of availability requirements 79

4.4 Verification of reliability requirements 81

5.1 Model construction statistics 96

LIST OF FIGURES

1.1 Proposed methodology 13

2.1 SEE in FPGAs [18] 19

2.2 Types of SEEs [99] 20

2.3 Probabilistic Model Checking 29

3.1 A sample pseudocode and DFG representation 33

3.2 Sample CDFG 35

3.3 CDFGs scheduled over available resources 35

3.4 Design option analysis methodology 37

3.5 Sample CTMC for reliability/availability analysis 41

3.6 PRISM modeling for a system with 2-adders and 2-multipliers 42

3.7 Safety modeling of simple system with safe and unsafe failure 43

3.8 PRISM modeling refined after inclusion of coverage *(c)* for a system with 2-adders and 2-multipliers 45

3.9 CDFG of an FIR filter 49

3.10 Failure probability vs I (scrub interval) 53

3.11 Reliability vs I (scrub interval) 54

3.12 Safety vs scrub I (scrub interval) 55

3.13 Impact of C (coverage) on the design with/without redundancy for I = 1) 56

3.14 Impact of C (coverage) on the design with/without redundancy for I = 9) 57

3.15 Impact of C (coverage) on for performability-area trade-off evaluation for I = 1 . . . 57

3.16 Impact of C (coverage) on for performability-area trade-off evaluation for I = 9 . . . 59

4.1 Design Assurance Levels . . . 63

4.2 Scrub optimization/DAL analysis methodology . . . 65

4.3 A simple Markov chain to illustrate failure occurrence . . . 67

4.4 State probabilities vs time . . . 68

4.5 Reliability vs time . . . 69

4.6 Markov chain for Erlang process . . . 69

4.7 Deterministic delay modeling with Erlang process . . . 70

4.8 Markov model of periodic blind scrub using Erlang process . . . 71

4.9 Conceptual TMR model . . . 72

4.10 Markov model of TMR with periodic blind scrub using the Erlang process 73

4.11 512-tap parallel FIR filter . . . 75

4.12 Availability for T = 300s . . . 77

4.13 Fault tree of the system . . . 80

5.1 Sample unmitigated circuit . . . 84

5.2 TMRed version of the sample circuit . . . 84

5.3 The sample circuit with TMR divided into two partitions . . . 84

5.4 Methodology for optimal TMR partitioning . . . 86

5.5 TMR implementation of the CDFG of an 8-tap FIR filter with two partitions . . . 88

5.6 Markov model of TMR with repair . . . 89

5.7 PRISM Code of a TMR with two partitions (SBU and DBU) . . . 91

5.8 SBU Markov model of TMR with two partitions 92
5.9 DBU Markov model of TMR with two partitions 93
5.10 SBU Reliability 97
5.11 SBU Availability 97
5.12 Combined Reliability 99
5.13 Combined Availability 99

LIST OF ACRONYMS

SEE	Single Event Effect
SEU	Single Event Upset
SEU	Single Bit Upset
MEU	Multi Bit Upset
CDFG	Control Data Flow Graph
SRAM	Static Random-Access Memory
FPGA	Filed Programable Gate Array
LUT	Look Up Table
TMR	Tripple Modular Redundancy
DAL	Design Assurance Level
PRISM	PRobabilistIc Symbolic Model checker
CTMC	Continuous-Time Markov Chain
DTMC	Discrete-Time Markov Chain
MDP	Markov Decision Process
PTA	Probabilistic Timed Automata
CSL	Continuous Stochastic Logic
CTL	Computational Tree Logic
HOL	Higher-Order Logic
DSP	Digital Signal Processing

Chapter 1

Introduction

1.1 Motivation

"In less than 70 hours, three astronauts will be launched on the flight of Apollo 8 from the Cape Kennedy Space Center on a research journey to circle the moon. This will involve known risks of great magnitude and probable risks which have not been foreseen. Apollo 8 has 5,600,000 parts and 1.5 million systems, subsystems and assemblies. With 99.9 percent reliability, we could expect 5,600 defects. Hence the striving for perfection and the use of redundancy which characterize the Apollo program."

— Jerome Lederer, Director of Manned Space Flight Safety, NASA, 1968.

The Avionics and space industries pose extra challenges to designers of electronic devices due to the critical nature of embedded electronic systems and their exposure to a radioactive environment that is significantly harsher than at sea level. It has been reported that airplane flying altitudes can have an environment that is 300 times [68] more radioactive than at sea level. High-energy neutrons caused by the interaction of

cosmic rays with the earths atmosphere may cause temporary or permanent failures in the electronic systems used in avionics and spacecrafts.

Field Programmable Gate Arrays (FPGAs) have been employed in aerospace applications for more than a decade. Reconfigurable computing with FPGAs is known to perform well in space-based applications such as Synthetic Aperture RADAR (SAR), software defined radio and hyperspectral imaging. With respect to power consumption and speed, FPGAs can outperform general-purpose CPUs. Also, due to field programmability, the absence of non-recurring engineering costs, low manufacturing costs and other advantages, SRAM-based FPGAs are increasingly attractive compared to ASICs. Since in SRAM-based FPGAs the configuration bitstream is stored in volatile SRAM, it can be corrupted by interaction with high-energy radiated particles such as protons, neutrons, and heavy ions that are abundant in aerospace environments. The effects of these particles on electronics are commonly known as single-event effects (SEE) [57, 5]. Several types of SEE are relevant to FPGAs. Single-event upsets (SEUs) occur when one or more bits in memory changes state due to a radiation event. Since the state of the FPGA configuration memory specifies the application architecture, SEUs in the configuration memory are particularly harmful to the system operations.

Different vendors have provided radiation-hardened FPGAs to meet the requirements of the avionic and space industries [88]. However, these devices are very expensive as they are manufactured in relatively low volumes and they also lag by two or three technology nodes when compared to commercial products. For example, if Xilinx 28-nm (non rad-hard) Virtex-7 (XC7V2000T) is compared to a Xilinx 65-nm rad-hard Virtex-5 (XQR5VFX130), the Virtex-7 offers 14.9 times more Configuration Logic Block (CLB) slices, 4.3 times more Block RAM memory and 6.8

Table 1.1: Comparison of FPGA Technologies [20]

Feature	**SRAM**	**Flash**	**Antifuse**
Reprogrammable	Yes	Yes	No
Volatile Configuration	Yes	No	No
Live On Startup	No	Yes	Yes
Memory Cell Size	Large	Small-Medium	Small
Radiation Sensitivity	High	Low-Medium	Low-None
Capacity	High	Medium	Low
Reprogramming Speed	Fast	Slow-Medium	N/A
Total Dose Tolerance	Medium-High	Low	High

times more Digital Signal Processing (DSP) slices running at twice the maximum frequency. Therefore, there is a growing need to analyze the possible utilization of commercial SRAM-based FPGA components in harsh radioactive environment such as outer space. Table 1.1 gives an overview of the features of the different FPGA technologies discussed, and is meant to provide a quick comparison of the main features, advantages and drawbacks. Here, capacity refers to the density and amount of logic that can be synthesised onto a single FPGA.

Dependability (reliability, availability and safety) and performability (reliability and performance combined) are major concerns in safety-critical and mission-critical applications that are common in the aerospace industry. To deal with SEUs, designers mostly rely on redundancy-based solutions, such as Triple Modular Redundancy (TMR) [21] for high reliability and *configuration memory (Configuration Bits) scrubbing* [5] to mitigate SEUs for high availability. Scrubbing is traditionally done in the order of milliseconds. Such fast scrubbing consumes high power [67, 59] and hence scrubbing at a lower frequency is desired [84]. Strict power budgets of typical deep space missions such as Voyager-1, Voyager-2 [98], or even the Mars missions set a need

for delayed and optimized scrubs (in the order of hours or days) to save power. Scrubbing is often used in conjunction with other forms of mitigation techniques such as TMR or spare components, to increase reliability. However, in cases where performability is a major concern, redundancy-based solutions might not always be the default choice. Thus, to choose the right design options and parameters, it is very important to evaluate the relationships between reliability, availability, safety, and performability with the adopted fault mitigation technique, fault coverage and mission time. To further increase dependability, TMR partitioning [75, 46] can be adopted. Although, assessing the optimal number of partitions at early design stages has yet to be addressed. Such early analysis will allow a designer to develop more reliable and efficient solutions, and may also reduce the overall cost associated with the design effort. Our work aims at achieving these goals.

Broadly, there are three possible ways to analyze SEU sensitivity in FPGA-based designs: 1) hardware testing such as particle beams and laser testing, which allows to obtain cross-sectional area information about the design; with the knowledge of flux, critical charge and charge collection efficiency, we can compute the SEU rate [38]; 2) fault injection emulation or simulation; and 3) analytical techniques. These three types of techniques are complementary, and they are typically applied at different design flow steps. Hardware testing techniques are the most realistic if the true operating conditions can be reproduced, which is not often the case. These techniques require finished implementations, and they may cause irreversible damage to the device under test when performed; therefore, they are very costly [58]. Some sufficient use of hardware testing techniques may, however, be mandatory for certification purposes in critical applications such as in aerospace electronic systems. Fault injection is also a useful method, but test time grows with the number of possible

test cases [14]. On the other hand, analytical methods tend to be relatively less accurate in some aspects. Nonetheless, they can provide much better controllability and observability, while enabling quick estimation of soft error susceptibility, without the risk of damaging devices [6, 86]. Moreover, they can capture features of the true test conditions that would be very hard to accurately reproduce when bombarding the circuit or while performing fault injection. Analytical estimation traditionally provides information at an earlier stage in the design cycle compared to the other two techniques.

We propose a means by which formal verification methods can be applied at early design stages to analyze the dependability and performability of reconfigurable systems. In particular, the focus is on *probabilistic model checking* [25]. It is used to verify systems whose behavior is stochastic in nature. Probabilistic model checking is a well known formal verification technique, mainly based on the construction and analysis of a probabilistic model, typically a Markov chain. The main advantage is that the analysis is exhaustive, which results in numerically exact answers to the temporal logic queries that contrast with discrete-event simulations [53], in which approximate results are generated by averaging results from a large number of random samples. Another extra advantage of this technique is its ability to express detailed temporal constraints on the system's executions in contrast to analytical methods.

There are numerous probabilistic model checking tools available based on numerical and statistical methods such as YMER [101], VESTA [81], MRMC [48] and PRISM [55]. In terms of memory consumption and performance, YMER is the best option [43]. Unfortunately YMER has a limited range of supported probabilistic operators (no unbounded until and steady-state operators). Furthermore, being a statistical model checker, YMER may report the wrong answer, and has done so in

a few cases [69]. YMER outperforms the other statistical model checker VESTA. VESTA's memory consumption is rather constant, but more in the order of PRISMs memory usage. However, its runtime varies considerably. For certain properties, it was found that VESTA does even terminate within 24 hours on a model with only 100 states [43]. While comparing the probabilistic model checking tools based on numerical methods, it is known that MRMC may outperform PRISM for small and medium-sized models in terms of speed and memory usage. PRISM is able to check much larger models when compared with the other numerical tools [69].

PRISM is an open source probabilistic model checker, and it also includes multiple model checking engines, several of which are based on symbolic implementations (using binary decision diagrams and their extensions, such as Multi-Terminal Binary Decision Diagrams). These engines enable probabilistic verification of models of comprising up to 10^{10} states (on average, PRISM handles models with up to $10^7 - 10^8$ states). PRISM also features a variety of advanced techniques such as abstraction refinement and symmetry reduction. It is worth mentioning that it also supports approximate/statistical model checking through a discrete event simulation engine. PRISM also offers excellent support via it's user group. In the PRISM model checker, probabilistic finite state models are constructed using real value probabilities associated with the transitions between various states of the model. It is known that probabilistic model checking tools run out of memory quickly when the state space is very large. In contrast, *probabilistic theorem proving* techniques [36], in theory, have no limitations regarding the number of states. However, they are interactive, which means that human interaction is required. Also, analyzing a system in a theorem proving environment requires an infrastructure for reasoning about the underlying mathematical concepts of probability and statistics.

In our proposed methodology, we are interested in addressing three issues in SRAM-based FPGA designs at early design stages: *Design options analysis*, *Design Assurance Level (DAL) verification with scrub optimization*, and *optimal TMR partitioning* for reliability improvement. Current work in the area of design options analysis [47, 62, 51] either separates the dependability analysis from the performance/area, coverage analysis or does not analyze such safety-critical applications at an early design stage. Commercial tools for reliability analysis, such as *isograph* [41], cannot be used for performability evaluation of such systems as they do not support Markov reward models [87]. Since the probabilistic model checker PRISM allows reward modeling, our work overcomes this limitation. Our contribution to optimize the scrub interval for saving power contrasts with other works in this area [22, 60], where the repair rates are estimated as exponential distributions. We model the deterministic repair intervals using the Erlang process [30] for better accuracy. Indeed, nonexponential holding time distributions can be approximated by inserting multiple intermediate states between every main state pairs. Our work on optimal TMR partitioning overcomes the limitations of the current approaches reported in [75, 46]. To be more precise, those approaches either employ the fault injection technique or are limited by the assumption of equal sized partitions prone to Single Bit Upsets (SEUs) only.

To analyze such a design at a high level using our methodology, we start from its Control Data Flow Graph (CDFG) [49] representation, obtained from a high-level description of the design expressed using a language such as C++. Depending on the desired type of analysis, the possible implementation options of the CDFG, with different sets of available components and their possible failures, fault recovery and repairs in the radiation environment are then modeled with the PRISM modeling language [76]. The failure rates of the components are obtained from a worst-case

component characterization library. Various dependability properties are then automatically verified to check if the system meets the requirements.

1.2 State-of-the-Art in SEU Analysis

The modeling and analysis of SEU relevant faults and their mitigation for dependability analysis is an active research area. As mentioned earlier, the most practiced approaches are fault injection and beam testing (radiation ground testing). Analytical methods were also proposed. Assadi et al. [7, 33] presented an analytical model based on soft-error propagation at gate level using the concept of Error Propagation Probability (EPP). Shazil et al. [82] used a similar concept, but used a satisfiability (SAT) solver with EPP to calculate the exact soft-error rate. The Probabilistic theorem proving technique was also employed in this area. Abbasi [4] extended the work of Hasan [36] by formalizing statistical properties of continuous random variables as well as the probability distribution properties of multiple random variables. Abbasi used this formalization for the formal reliability analysis of engineering systems using theorem proving. On the other hand, in our work, we focus on Markov modeling of SEU impacts and their possible mitigation techniques. Such modeling enables evaluating the dependability of a system at the early design stages from its CDFG description using the probabilistic model checking technique. The state-of-the-art literature that are most relevant to the three main parts of our methodology are described as follows.

1.2.1 Modeling and Analysis of Fault Mitigation Techniques

The modeling and analysis of different fault mitigation techniques, for both ASICs and FPGAs, has been widely reported [16, 39, 45, 91, 9]. When a resource fails (due to a configuration bit flip), an alternative CDFG scheduling using high-level synthesis

techniques can be derived to continue the system's operation using the remaining resources, most likely at a lower throughput. Such a fault tolerance approach was introduced by Borgerson et al., Hong et al. and Karri et al. in [16, 39, 45], respectively, for fault-secure microarchitectures and multiprocessors (a computation on a set of processors is fault-secure if no fault in the computation generated by a faulty processor goes undetected). For FPGA-based designs, such a fault recovery technique can be adopted as well. However, in that case the controller for rescheduling the operations will need to be highly reliable. This controller can be implemented in a separate chip with proper fault-tolerance mechanisms.

In [91], Tosun et al. proposed a reliability-centric high-level synthesis approach to address SEUs. Their framework uses reliability characterization to select the most reliable implementation for each operation fulfilling latency and area constraints. In addition, researchers dedicated a lot of efforts to modeling the behavior of gracefully degradable large-scale systems using continuous-time Markov reward models [9, 40]. In [24], Cheshmikhani et al. presented the modeling and analysis of a fault tree based on stochastic logic. To produce models, probabilistic analysis of all different types of gates is carried out first, and then probabilistic models are converted to equivalent stochastic logic gates. It is worth mentioning that unlike Markov chains, a classical fault tree is limited to modeling only non-repairable systems.

Fault coverage has a considerable impact on systems' reliability and safety, and many papers reported approaches for safety modeling and dependability improvement mostly based on improving the fault detection coverage. The impact of coverage on reliability with a quantitative assessment of different types of systems were performed and reported by Xing et al., Delong et al. and Verlinden et al. in [100, 31, 95]. Always setting a target of 100% coverage is expensive in terms of time and cost, as

well as unnecessary in many cases. Unfortunately, none of these works address the relationship between fault coverage, different fault mitigation techniques and mission time for early design analysis.

Smith et al. presented a case study in [83] to measure the performance of a multiprocessor system using a continuous-time Markov reward model (MRM). Similar to this work, Kumar et al. [51] presented another MRM-based approach for analyzing the performance, area and reliability metric of a design. In this work the transistor lifetime was used to model the reliability and performance; hence, the model is composed of non-repairable modules. The use of a non-formal commercial tool (such as the *isograph* [41]) makes their approach quite rigid in terms of analysis. Moreover, in their proposed approach, reward calculation is manual, which is a major bottleneck of their work.

1.2.2 TMR and Scrub Modeling

Researchers have put a lot of efforts into hardening SRAM-based FPGA designs using TMR and scrubbing. In [15], Bolchini et al. presented a design flow to implement SEU hardened systems implemented with SRAM-based FPGAs. Three independent strategies were proposed. The proposed strategies are the TMR-based techniques, the TMR coupled with partial reconfiguration and some specific local re-design of the critical portion of the design to overcome TMR failures. Quinn et al. [78] presented a number of possible radiation-induced faults in SRAM-based FPGAs and their mitigation methods. However, they did not provide any model for the estimation of the rate at which those errors happen and how to handle them at an early design stage. An important class of applications with deterministic maintenance and repair times was proposed by Trivedi et al. in [23]. Their work presented a steady-state analysis

of the periodic preventive maintenance problem with general failure and repair time distributions obtained by solving a semi-Markov process.

All these works mentioned above either use semi-Markov models or assume a time interval between scrubs to follow exponential distribution, which is not accurate in real world scenarios. Even though semi-Markov processes have been employed to model deteriorating systems by allowing the holding time distributions to be non-exponential, it is assumed that the mathematical formulations of semi-Markov models [29] are so complicated that they are not analytically tractable. Some of the works mentioned above are mostly focused on designing a more robust TMR solution and none explored the effect of scrub intervals on the FPGA dependability. In our work, we are interested in using formal verification techniques as they can guarantee exact solutions. We model the periodic scrub using conventional Markov chains. Nonexponential holding time distributions in Markov chains are approximated by inserting multiple intermediate states based on a phase-type distribution.

1.2.3 TMR partitioning

Reliability and availability predictions of TMR have been heavily studied recently. Tambara et al. [89] reported the effectiveness of different TMR schemes implemented with a different level of granularity after evaluating them experimentally (using beam testing). In contrast, Wang et al. [97] proposed an analytical model for systems with TMR, TMR with EDAC and TMR with scrubbing. The authors discussed Markov modeling of these techniques; however, frequent voting or partitioning was not addressed. Sterpore et al. presented an interesting scrubbing approach for TMRed designs in [85]. They presented a design flow to scrub each domain in a TMR independently to maximize availability. In this approach, each partition is scrubbed

on-demand when required. Since TMR is very expensive in terms of area and power, another interesting way of implementing TMR, known as "selective TMR" was introduced by Pratt et al. in [74]. In their work, they showed how TMR can be applied only on selected portions of a design to reduce cost. Even though some level of reliability is sacrificed in this approach, in terms of area constraint, their tool maximizes the reliability.

TMR partitioning was mainly addressed in two papers. Pratt et al. [75] proposed a reliability model for partitioned TMR, but only for designs with equal sized partitions. Lima et al. [46] demonstrated the effect of Domain Crossing Events (DCEs) and how to insert the voter cleverly in a design. In their work, they analyzed different partitioning schemes for the same design, and using the fault injection technique, they find the optimal number of TMR partitions suitable for that design. Our work contrasts with all of these related works mentioned above. We focus mostly on the modeling of both Single Bit Upsets (SBUs) and Multi Bit Upsets (MBUs) for early analysis of a design. The proposed modeling can handle both equal and not equal sized partitions. Apart from just measuring the dependability metrics, such a methodology can help in analyzing the relationship between the number of partitions, the scrub interval, and the mission time at early design stages to improve confidence in a design.

1.3 Proposed Methodology

In Figure 1.1, we present the proposed methodology. We start from the CDFG of the application extracted from the high-level description of the design. The CDFG model is used to represent the functionality of the selected C/C++ modeled algorithm. Such a CDFG model is composed of structures for arithmetic or logical operations, and is

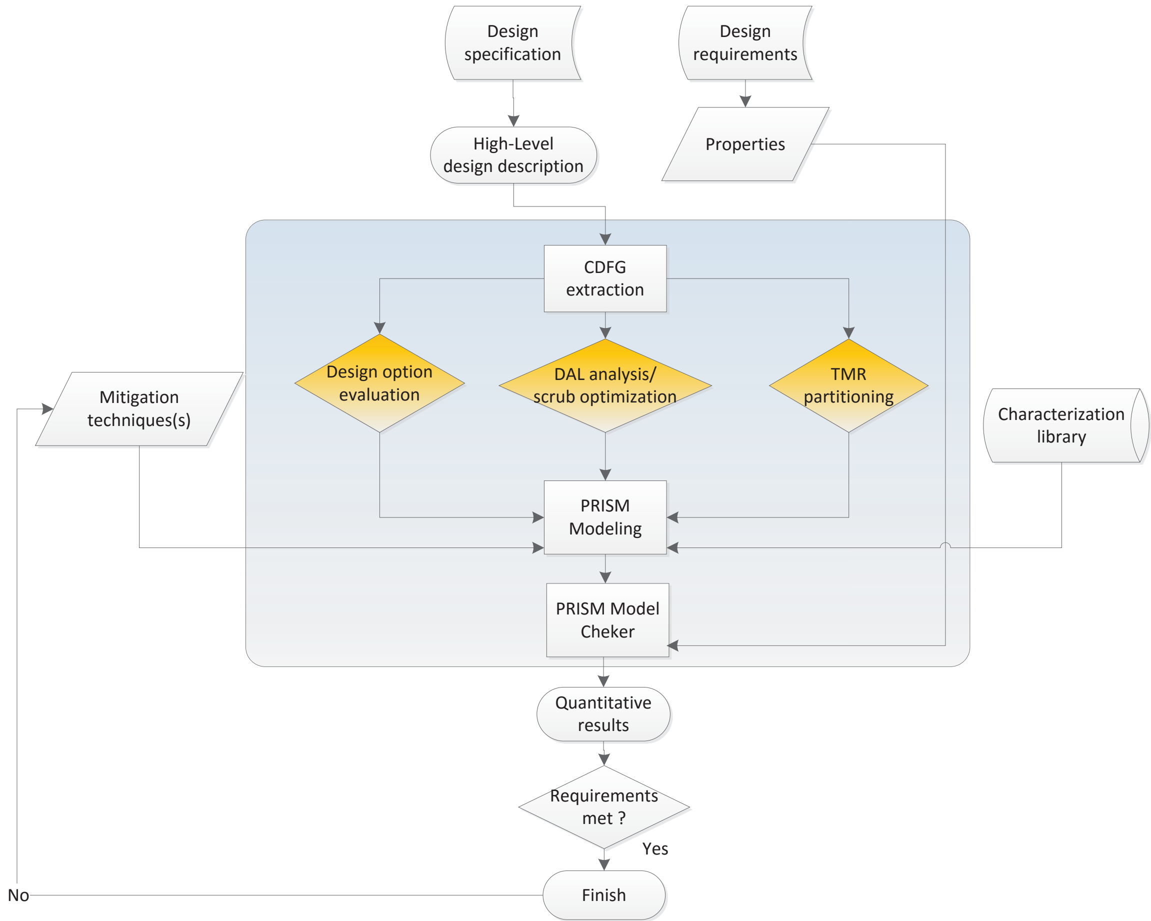

Figure 1.1: Proposed methodology

also capable of representing all behaviors present in an algorithm. Different tools such as GAUT [28], SUIF [32], etc. can be used to extract the CDFG from a high-level design description expressed using a language (such as C/C++). C/C++ is the dominant high-level language in embedded system programming. However, the approach can be applied equally well to other languages, such as Java or an older language like Fortran. The C/C++ model can be written by the designer or can be generated automatically in case the design is modeled in Matlab/Simulink. The embedded coder (formerly RTW) toolbox can be used to perform the transformation. The idea of the *CDFG extraction* and *characterization library* is inspired by [90];

however, we use the GAUT tool as it is a free tool for academic use that is also user-friendly. We also developed a version of our *characterization library* [Bio-Cf5] using the information in [90] to calculate their SEU rate in the higher earth orbit.

Once the CDFG is extracted, depending on the type of analysis desired (showed as orange boxes), one of the three branches can be chosen in our methodology. Each of the orange boxes has its own flow that will be described in detail in the relevant chapters. The *design option analysis* focuses on the evaluation of available design options and the use of rescheduling [16] for optimization of reliability and performance. The *scrub optimization/DO-254 Analysis* emphasizes the accurate modeling and optimization of scrubbing for saving power. This branch of the methodology can also be used to verify if the design meets the Design Assurance Level (DAL) described in DO-254 [65]. The optimal approach for partitioning a TMRed system is analyzed in the *TMR partitioning* branch. For this purpose, we model the effect of single and multiple-bit upsets on the design that enable us to quantitatively analyze the number of TMR partitions with the adequate scrub frequency required for the design.

All of these analyses require the use of a *characterization library* that characterizes the failure rate and area for various components such as different types of adders, multipliers, and so on. Each node in the CDFG defining basic operations can be directly mapped into one of these components in the library. The details of this *characterization library* development will be discussed in chapter 3 of this thesis. In our work, we use the first-order worst case scenario for the development of the *characterization library*. Note that we use the *characterization library* to obtain the failure rate of the components in the Markov chain model and the methodology is generic enough to be used with a different characterization library with more precise and accurate data, without any significant changes.

The modeling of these three types of analysis in our methodology is based on either Continuous Time Markov Chains or Markov Reward Models. Once described in the PRISM modeling language [54], the PRISM model checker automatically analyzes the dependability properties. In the case of stochastic modeling, properties are usually expressed in some form of extended temporal logic such as the Continuous Stochastic Logic (CSL) [37]. CSL is a stochastic variant of the well-known Computational Tree Logic (CTL) [25]. If the system does not meet the requirements, then the mitigation approach is modified, and the analysis is repeated again.

1.4 Thesis Contributions

The main contribution of this thesis is the methodology based on the probabilistic model checking technique to qualitatively analyze the dependability of SRAM FPGA-based aerospace applications. Since fault injection and beam testing requires the finished implementation of the design and comes with an associated cost and high risk, our methodology can be used for early dependability evaluation. This will guide designers to adopt the proper mitigation technique or the combination of mitigation techniques to make a more reliable, efficient and robust design. This may save on design effort, design time and may also reduce the overall cost of the product. We list below the main contributions of this work with references to related publications provided in the Biography section at the end of this thesis.

1. We developed Markov Reward Models to analyze design options with respect to reliability, availability, safety, and performability-area tradeoff for early design decisions. The quantitative results from our obtained model show some important observations such as the fact that high coverage is not always helpful for gaining high reliability, and that scrubbing delay also has a considerable impact. Regarding safety,

using our models, we demonstrate how the scrubbing interval affects the safety of available design options with the same fault coverage. We prove that in some cases, redundancy-based solutions might not be always the best choice as one may expect. Alternatively, for those cases, rescheduling in conjunction with scrubbing can be a good option. To our knowledge, this is the first attempt to evaluate such relationships at early design stages using probabilistic model checking [Bio-Jr2, Bio-Cf5].

2. We developed the Markov model for reconfigurable systems with periodic scrub and TMR using conventional Markov chains (instead of semi-Markov chains). This enables us to assess the system using a probabilistic model checker in order to optimize the scrub interval. Using the same model, we also verify if a system meets the design assurance level to comply with the DO-254 and also high-availability requirements. Unlike the traditional approach where the repair rates are estimated as exponential distributions, we model the deterministic repair intervals using the Erlang process [30] for better accuracy [Bio-Cf3, Bio-Cf4].

3. To analyze the TMR partitioning at an early design stage, we developed Markov models that can handle both the equal sized or non-equal sized partitions. Besides, these models can quantitatively assess the effect of both SBUs and MBUs on TMR partitions. Our analysis shows that increasing the number of TMR partitions also increases the design reliability for the case of SBUs. In contrast, for designs that are prone to both SBUs and MBUs an optimal number of partitions indeed exists. We prove that with an increased number of partitions, less frequent scrub will be required to meet a target reliability. On the other hand, a smaller number of partitions will require more frequent scrubs. Using our methodology, it is possible to assess the required scrub frequency and the number of TMR partitions at early design stages for a specific mission time. [Bio-Jr3]

1.5 Thesis Organization

The rest of the thesis is organized as follows: In Chapter 2, we provide a brief overview of SEUs and configuration scrubbing. We also provide in this chapter an introduction to probabilistic model checking, the PRISM model checker and some basic definitions for some relevant dependability metrics.

In Chapter 3, we present the reliability, availability, safety, and performability modeling of design options using Markov Reward Models (MRMs). We also present modeling results to support our observations.

The modeling with Erlang process for approximating a constant time delay to optimize the scrub interval is discussed in Chapter 3. In this chapter, we also show with a case study how the proposed models can be used for verification of DAL compliance.

In Chapter 4, we present the details of modeling the SBU and MBU impacts in a partitioned TMR system. We analyze a system for both the SEUs and MBUs, and quantitatively analyze the required number of portions with proper scrub frequency.

Finally, Chapter 5 provides concluding remarks and several future research directions.

Chapter 2

Preliminaries

In this chapter, we provide a brief overview of the concepts required to understand this thesis. We start by introducing the Single Event Effects (SEEs) and the different types that may occur in FPGAs. Since this thesis mostly focuses on Single Event Upsets (SEUs), the consequences of SEUs FPGAs are also be introduced in this chapter. In addition, we also provide the detail of fault mitigation techniques such as Triple Modular Redundancy (TMR) and scrubbing of configuration bitstreams. We also provide definitions of various dependability metrics that will be assessed in this thesis. We conclude this chapter by providing a short overview of the probabilistic model checking — the formal verification technique that we used in our methodology.

2.1 Single Event Effects

The harsh radioactive environment in space may affect the sensitive electronic systems. As shown in Figure 2.1, when charge from radiation particles is impinging on a device, it has the potential of altering the internal state of, or damaging, the device. Such incidents are known as Single Event Effects (SEEs).

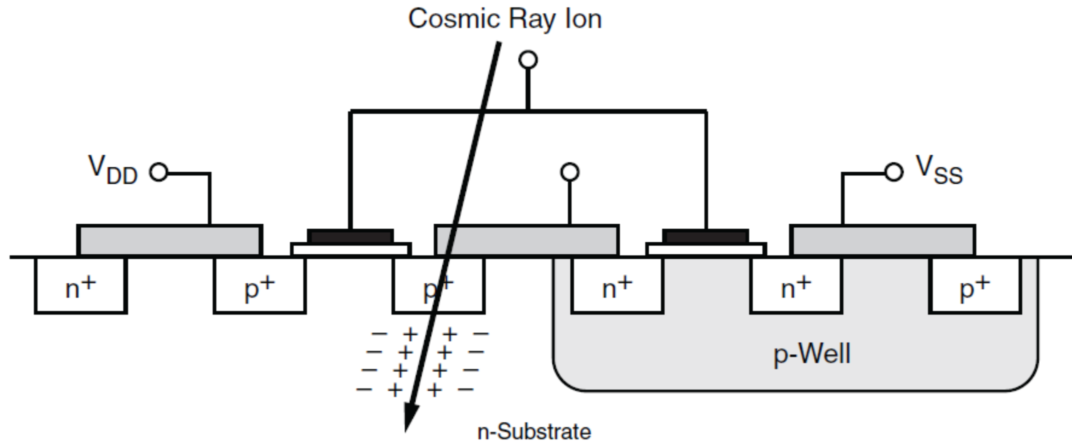

Figure 2.1: SEE in FPGAs [18]

As shown in Figure 2.2, different types of SEEs that can occur in FPGAs can be divided into *soft errors* and *hard errors*. We provide the summary of different types of SEEs as follows, however, in this thesis, we deal with the Single Event Upsets only.

2.1.1 Soft Errors

Soft errors (recoverable) are upsets to the device operation and are self-correcting in time or are correctable by rewriting a memory element. The three subclasses of soft errors are Single-event transients (SETs), Single-Event Upsets (SEUs) and Single-Event Functional Interrupts (SEFIs) that can be described as follows:

1. **Single-Event Transients (SETs)** result when a high-energy particle impacts a combinatorial path of a device and can induce a voltage/current spike. If the pulse-width of this spike is sufficient and arrives at the right time, it can propagate through the circuit to a state flip-flop or latch.

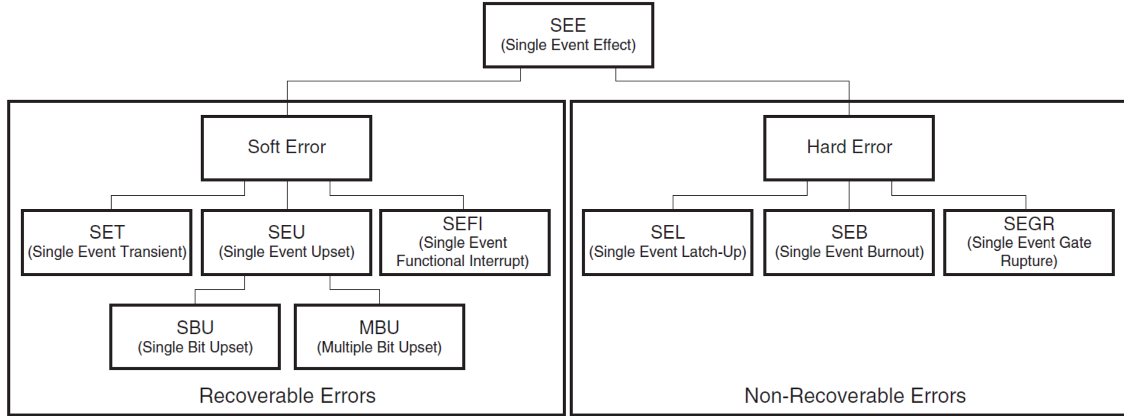

Figure 2.2: Types of SEEs [99]

2. **Single-Event Upsets (SEUs)** are the result of high-energy particles causing a change in the state of a memory element (SRAM, flash, flop, or latch). SEUs can be categorized as single-bit or multi-bit upsets (SBUs or MBUs). SBUs are by far the most common SEE seen in avionics applications [99].

3. **Single-Event Functional Interrupts (SEFIs)** are disruptions to normal device operation. These types of effects alter the functionality of the circuit and typically require reconfiguration/reset or power cycling for recovery.

2.1.2 Hard Errors

Errors that cause lasting damages to devices are classified as hard errors (non-recoverable). The three subclasses of hard errors are Single-event latch-up (SEL), Single-event burnout (SEB) and Single-event gate rupture (SEGR) described as follows:

1. **Single-Event Latch-up (SEL)** is a circuit latch-up induced by radiation. This latch-up can be either permanent or clearable with power cycling.

2. **Single-Event Burnout (SEB)** is a short-circuiting caused when a high-energy ion impacts a transistor source, causing forward biasing. SEBs are typically a threat to power MOSFETs but are also seen in IGBTs, high-voltage diodes, and similar circuits.

3. **Single-Event Gate Rupture (SEGR)** is a plasma spiked caused by a high-energy ion impact, resulting in rupture of the gate oxide insulation.

2.2 Single Event Upsets in SRAM-based FPGAs

FPGAs are configurable logic devices that implement logic circuits with a fabric that includes Look-up tables (LUTs), memories and routing resources that connect the LUTs and memories. LUTs are used to implement logic equations. In contrast, memories are used for implementing sequential logic and storage. In a reconfigurable FPGA, the configuration memory is a collection of static memory cells storing the bitstream. Bitstream bits set the values of the LUTs, flip-flops and memory initialization values, and states of switches and connection boxes that route signals through the FPGA. For Virtex devices from Xilinx, the configuration memory is composed of SRAM cells. Those cells are arranged in frames of 32-bit configuration words. In Virtex-5, there are 41 words in each frame [1].

Several interfaces are provided for accessing configuration memory for different purposes. The Joint Test Action Group (JTAG) interface is typically used for initial configuration. The Xilinx-specific SelectMAP interface is used for runtime read-back and reconfiguration. It can be configured for a bus width of 8, 16, or 32 bits. Xilinx provides the Internal Configuration Access Port (ICAP) to expose the SelectMAP interface to user logic. The ICAP eliminates the need for an external runtime-configuration manager by allowing the FPGA to read back and reconfigure

itself.

The FPGA configuration is stored in volatile SRAMs. Therefore, interaction with high-energy radiated particles that are common in the aerospace environment, such as protons, neutrons, and heavy ions, may corrupt the FPGA configuration. Single-Event Upsets (SEUs) occur when one or more bits in configuration memories change state due to a radiation event. If only one bit of a word is affected, then it is called a Single-Bit Upset (SBU). If more than one bit are affected, then it is an MBU. The state of the FPGA configuration memory defines the architecture of the application. As a consequence, SEUs in the configuration memory are not only harmful but could also result in a catastrophic failure of the design. Many bits in the bitstream that are not employed in a given design do not affect system operation if an upset occurs on them. A portion of the configuration memory bits that are employed in the design directly affects the system operation if an upset occurs on them, and these critical bits can only be identified by fault-injection techniques. However, for estimation, Xilinx allows generation of a mask file that identifies the essential bits of the design, of which the critical bits are a subset [56].

Altera also offers a variety of SRAM-based FPGAs, such as the Stratix and Cyclone devices [2]. However, Altera FPGAs mostly rely on CRC-based error detection and correction methodology for SEU mitigation [3]. If there was a means of estimating the number of essential bits and of enabling the periodic scrubbing technique, our proposed methodology would apply directly on Altera SRAM-based FPGAs as well.

2.3 Scrubbing

Data scrubbing is a well known technique for error correction. It uses a background task that periodically inspects memory for errors and corrects the error using error-correcting code memory or redundant copy of data. Scrubbing in Xilinx FPGAs use an approach to scrubbing for configuration memory.

For most applications, the FPGA configuration data is loaded upon power-up. In such applications, the desired state of the configuration memory that enables the repair of upset bits through scrubbing will be known. For the implementation of the scrubber, there are mainly two options. The first option is to implement the scrubber as an external device, such as a radiation-hardened microprocessor. The other option is to implement the scrubber internal to the FPGA using the fabric and ICAP [13]. External scrubbing with radiation-hardened parts is reliable. However, as it requires at least one additional processor, it can be expensive in terms of power, size, and cost. Even though internal scrubbing is superior with respect to these constraints, extra care is required for implementing internal scrubbing as the scrubber itself is vulnerable to SEUs. A scrubbing technique is a single algorithm used in the system to mitigate configuration-memory upsets. There are two different types of techniques, specifically detection techniques and correction techniques. Each of these techniques has its own properties with respect to error coding, granularity and redundant data sources. A scrubbing strategy is composed of at least one correction technique and optionally, a detection technique. Blind scrubbing is a very popular scrubbing strategy with no detection technique. If at least one detection technique is used in a scrubbing strategy, then it is called read-back scrubbing. In read-back scrubbing, the current state of configuration memory is read back from the device to detect an upset. For this thesis, we concentrate on blind scrub, that does not require any detection.

As discussed above, all scrubbing strategies employ at least one technique for correcting the configuration bit upsets. Correction techniques either use data redundancy to recall or calculate the original configuration and then write this configuration to the device. These techniques differ in their coverage of various upset types (e.g., SBU vs. MBU), granularity of correction (e.g., frame vs. device) and correction data source (e.g., off-chip vs. on-chip memory). Depending on the chosen scrubbing strategy, the correction technique may be triggered continuously by a simple timer delay, or by a detection technique. The golden copy correction and error syndrome correction are the two main correction techniques that are widely used for scrubbing. Error syndrome correction is mostly used in read-back scrubbing. In golden copy correction, a trusted golden copy of the original configuration is kept off-chip in non-volatile storage, such as a radiation-hardened PROM, and used to reconfigure the FPGA as needed. Blind scrubbing strategies that employ only golden copy correction are very popular in FPGA-based space platforms because of their effectiveness (they can fix any number of upsets) and simplicity (less implementation complexity). These strategies continuously or periodically reconfigure the FPGA with the golden copy to repair errors quickly after they occur. A known limitation of blind scrub is that the radiation-hardened memories may have limited bandwidth. As a result, the configuration clock often can not be run at its maximum frequency. Scrubbing can be done at a specified rate meaning that there might be a period of time between the moment the upset occurs and the moment when it is repaired. That is why another form of mitigation is required, such as a redundancy-based solution known as TMR [21]. TMR is a technique for enhancing the reliability, in which each module in a circuit or the whole system is triplicated. A majority vote (two out of three) is taken on the TMR outputs to determine the final module output.

A scrub rate describes how often a scrub cycle should occur. It is denoted by either a unit of time between scrubs, or a percentage (scrub cycle time divided by the time between scrubs). There are direct relationships between scrub rate, design size, design reliability and design safety, hence the scrub rate should be determined by the expected upset rate of the device for the given application.

2.4 Dependability Metrics

The term dependability has quite a broad meaning that varies in the literature. In 1988, a survey [71] on several definitions of computer-based system dependability resulted in this concise definition: *"Dependability of a computer system may be defined as justifiable confidence that it will perform specified actions or deliver specified results in a trustworthy and timely manner"*.

Similar to the definition, the number of dependability attributes also has several options. According to [71], the attributes are : reliability, availability, performance, integrity, robustness, serviceability, resilience, maintainability, testability, safety and security. Some of the selected attributes will be discussed in the following as those attributes will be analyzed using our proposed methodology.

2.4.1 Reliability

Reliability of a system (or component) is defined as the probability that the system performs correctly for a given period of time, from zero (t_0) to t_1, given that the system or the component was functioning correctly at t_0. The reliability $R(t)$ of a single system/component (non-redundant) can be expressed as

$$R(t) = e^{(-\lambda t)} \tag{2.1}$$

where λ represents the failure rate of the component or the system, and t represents the time period.

2.4.2 Availability

Availability is defined as the ratio of time the system or component operates correctly (system uptime) to its entire mission time. For a simple system, if the Mean Time Between Failure (MTBF) and the Mean Time To Repair (MTTR) are known, then the availability of the system or the component can be expressed as

$$A = \frac{MTBF}{(MTBF + MTTR)} \tag{2.2}$$

More generally, Operational availability A_o [44] is expressed as

$$A_o = \frac{Uptime}{(Uptime + Downtime)} \tag{2.3}$$

2.4.3 Safety

Safety can be defined as a probability $S(t)$, which represents that the system either behaves correctly or will discontinue its function in a manner that causes no harm (operational or fail-safe). The notion of *coverage* is very important to model safety using Markov chain. The *coverage* is the measure of the system's ability to reach a fail-safe state after a fault. Modeling coverage and safety in Markov chain means that every 'unfailed' state has two transitions, to a fail-safe and to a fail-unsafe state.

2.4.4 Performance

Performance can be considered as the sub-characteristics of dependability. Some of the most common performance metrics are response time, throughput and resource utilization. Throughput is a more popular metric than response time to compare the performance of a system. According to ISO 9126-2, throughput describes the amount of tasks that can be performed over a given period of time. Throughput can be measured in different ways, depending on the system and devices involved to handle the task. For example, for a batch system, throughput is measured in job/sec. in contrast, in interactive systems, request/sec is used.

2.4.5 Performability

Performability metric quantifies the system's ability to perform in the presence of faults [83]. It combines the performance and reliability of a fault tolerant system to quantify the operational quality of the service between the failure (or an error) and its recovery. The results of performability evaluation can be used as a supplement to other metrics to assess the trustworthiness of a system. The concept of "performability" was first proposed in the mid-1970s, when system designers started dealing with large and complex systems that need to maintain some degree of functionality after a fault [64]. The field of fault-tolerant computing uses performability metrics extensively to provide a composite measure of both performance and reliability over the entire lifetime of a system [19]. The fault tolerant computing field frequently deals with safety-critical and mission-critical systems for demanding environments such as aerospace and high-performance transaction processing. Due to the high cost of developing and servicing of such systems, designers strive to model the behavior of such a

system before they actually build it. Performability metrics provide a way to quantify the behavior of such systems to gain more confidence in their design in early stage.

To get more familiar with the metric performability, consider a remote sensing satellite (for weather forecasting) that is to be launched into orbit. The satellite consists of a set of redundant components for its main crucial parts, such as high-definition cameras, antennas and its power distribution system. Each of these component has an associated failure rate over time, as well as a performance contribution. The performance contribution of a component defines the degree to which the overall performance of the satellite will be degraded upon the failure of that component. Given these information, system designers can build an analytical model to determine the average level of performance that the satellite can be expected to provide for part or all of its operational lifetime. This metric, which might be a throughput measurement, such as images transmitted per day (or any other number of specific tasks completed per second/hour/day etc.), is a performability metric because it captures the system's behavior given a failure (or fault) condition.

One way to measure the consequence of system's performance degradation due to a failure is to reward the system for every time unit it is ready to perform its task, at a rate which is proportional to its performance during this interval. This can be achieved by *Markov Reward Model (MRM)*, which will be discussed in more detail later in this thesis.

2.5 Probabilistic Model Checking and PRISM

Model checking [25] is a well established formal verification technique to verify the correctness of finite-state systems. Given a formal model of the system to be verified in terms of labelled state transitions and the properties to be verified in terms of

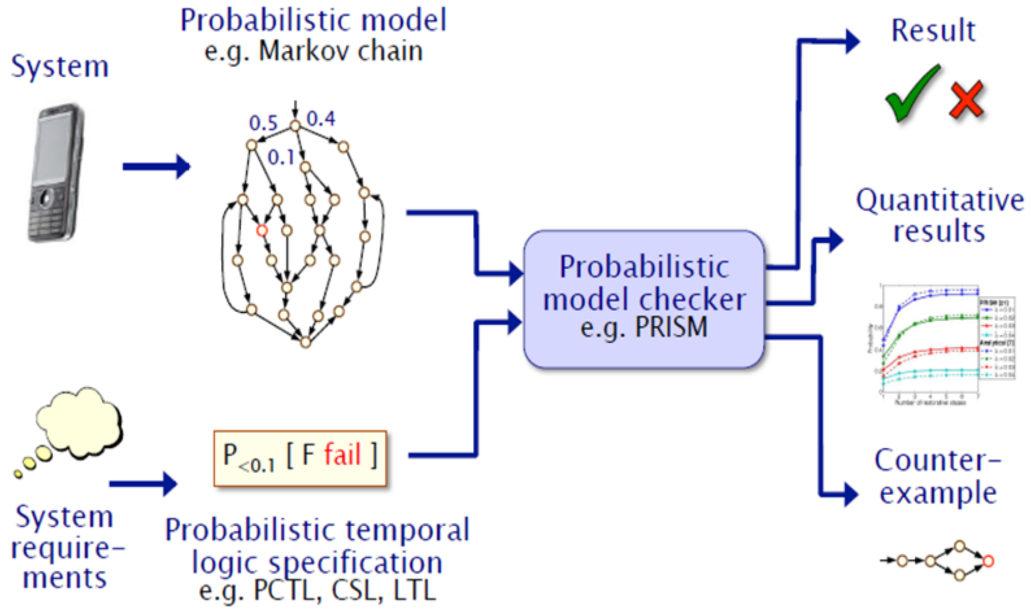

Figure 2.3: Probabilistic Model Checking

temporal logic, the model checking algorithm exhaustively and automatically explores all the possible states in a system to verify if the property is satisfiable or not [26]. If not, a counterexample is generated. *Probabilistic model checking* deals with systems that exhibit stochastic behaviour, such as fault-tolerant systems. Probabilistic model checking is based on the construction and analysis of a probabilistic model of the system, typically a Markov chain. In this thesis, we focus on the use of continuous-time Markov chains (CTMCs) and Markov reward models [87], widely used for reliability and performance analysis.

A CTMC comprises a set of states S and a transition rate matrix $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$. The rate $\mathbf{R}(s, s')$ defines the delay before which a transition between states s and s' takes place. If $\mathbf{R}(s, s') \neq 0$ then the probability that a transition between the states s and s' might take place within time t can be defined as $1 - e^{-\mathbf{R}(s,s') \times t}$. No transitions will take place if $\mathbf{R}(s, s') = 0$. Exponentially distributed delays are suitable for modelling component lifetimes and inter-arrival times.

In the model-checking approach to performance and dependability analysis, a model of the system under consideration is required together with a desired property or performance/dependability measure. In case of stochastic modelling, such models are typically CTMCs, while properties are usually expressed in some form of extended temporal logic such as Continuous Stochastic Logic (CSL) [8], a stochastic variant of the well-known Computational Tree Logic (CTL) [25]. PRISM [55] (as shown in Figure 2.3) is a well known tool for the formal modeling and verification of stochastic systems, currently supports four types of probabilistic models: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), discrete-time Markov decision processes (MDPs) and probabilistic timed automata (PTA). The specification language for properties of the probabilistic models to be analysed in PRISM is based on temporal logic, in particular PCTL and CSL that are probabilistic extensions of the logic CTL. The principal operators are P, S and R which refer, respectively, to the probability of an event occurring, the long-run probability of some condition being satisfied and the expected value of the models costs or rewards. Below are given two illustrative examples with their natural language translation in PRISM:

1. $P\ =\ ?\ [\ F[0,600]\ fail_A\ ]$ - "The probability that component A fails within 10 minutes".

2. $P\ =\ ?\ [\ G[0,3600]\ !(fail_A|fail_B)\ ]$ - "The probability of no failures occurring in the first hour".

Additional properties can be specified by adding the notion of rewards. Each state (and/or transition) of the model is assigned a real-valued reward, allowing

queries such as:

$R\{``oper"\} = ?[\,C < T\,]$ - "The expected cumulative operational time of the system in the time interval [0, T]".

Rewards can be used to specify a wide range of measures of interest, for example, the number of correctly delivered packets or the time that the system is operational. Of course, conversely, the rewards can be considered as costs, such as power consumption, expected number of failures, etc.

Chapter 3

Design Option Analysis

In safety-critical applications, dependability and performability of a design are prime concerns since failures may have catastrophic consequences. This sets a need to analyze different design options with different mitigation technique(s) for early design decisions. In this chapter, we will discuss details of the *design option analysis* part of our proposed methodology. Configuration scrubbing is often used in conjunction with other forms of mitigation techniques such as TMR or spare components, to increase reliability. However, cases where performability (reliability and performance combined) is a major concern, redundancy-based solutions might not always be the default choice [Bio-Cf5]. Moreover, setting always a target of perfect coverage is expensive and unnecessary in most cases. That is why, it is crucial to evaluate the relationship between reliability, availability, safety and performability with the adopted fault mitigation technique(s), fault coverage and mission time. Such analysis at an early design stage will allow designers to develop more reliable and efficient solutions, and may also reduce the overall cost associated with the design effort. This part of our work aims at achieving these goals.

Starting from a high-level description of the design, a Markov reward model is

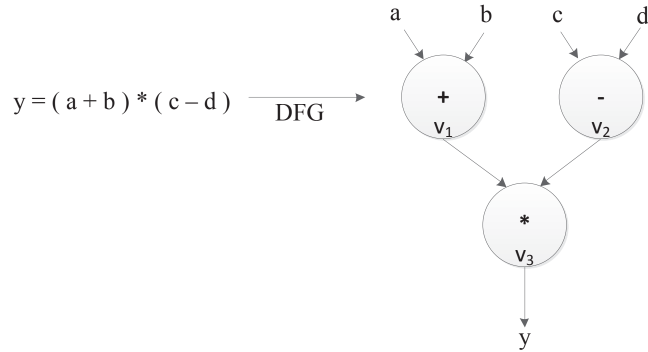

Figure 3.1: A sample pseudocode and DFG representation

constructed using the Control Data Flow Graph (CDFG) of the system and a component characterization library targeting FPGAs. The proposed model and exhaustive analysis captures all the failure states, fault detection coverage and repairs possible in the system within the radiation environment. In this chapter, we will also present quantitative results based on an FIR filter circuit to illustrate the applicability of the proposed approach.

3.1 CDFG and High-level Synthesis

In high-level synthesis, a behavioral description of a system is transformed into a structural description comprising data path logic and control logic. High-level synthesis algorithms read a high-level description and translate it into an intermediate form. An intermediate form should represent all the necessary information and be simple to be applicable in high-level synthesis. A Control Data Flow Graph (CDFG) refers to such an intermediate form. At first, a behavioral description is converted into a CDFG. The operations of the CDFG are then scheduled in clock cycles (scheduling), a hardware module is assigned to each operation (module assignment), and registers are assigned to input and output variables (register assignment). A Data Flow Graph

(DFG), in which control information is omitted from a CDFG, is used frequently for data path-intensive circuits (to which high-level synthesis applies) such as filters and signal processing applications. They often do not require control of the data flow. In Figure 3.1, a sample pseudocode (on the left) and its equivalent DFG representation (on the right) are shown. There are control-intensive circuits in which control of the data flow is required. The control data flow graph representation is used for such circuits. Since our work mostly focuses on data path-intensive applications, we utilize the information about the Data Flow (DFG) part of a CDFG. A formal definition of a DFG can be given as: *A DFG is a directed graph $G = (V, E)$, where $V = v_1, v_2, ..., v_n$ is a finite set whose elements are "nodes" and $E = V \times V$ is an asymmetric "dataflow relation", whose elements are called "data edges".*

Scheduling is an important part of high-level synthesis. Scheduling can be described as the process of dividing the CDFG (or DFG) into time steps that corresponds to clock cycles at the Register-Transfer Level (RTL) level. A small example is shown in Figure 3.1 that can be scheduled in two control steps (time steps) using two adders and one multiplier. However, it can also be scheduled in three control steps using only one adder and one multiplier. High-level synthesis algorithms such as forced-directed list scheduling (FDLS) [73, 94] can generate different CDFGs depending on component availability. FDLS is a well-established resource-constrained scheduling algorithm in high-level synthesis of digital circuits that utilizes the strengths of the Force-Directed Scheduling (FDS) and List Scheduling (LS). Force-Directed List Scheduling takes the resource constraints and tries to optimize the latency of the design. FDLS is similar to LS except for the force (measure of concurrency) used as a priority function. In brief, FDLS maintains a priority list of operations at each time step. From this, the FDLS

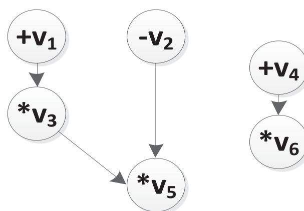

Figure 3.2: Sample CDFG

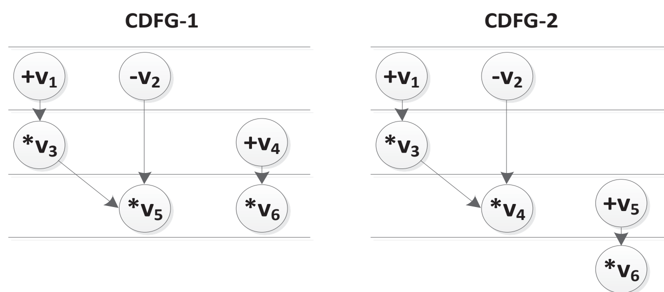

Figure 3.3: CDFGs scheduled over available resources

algorithm schedules operations until the resources become insufficient, and defers the rest.

3.2 CDFG Rescheduling for Fault Recovery

Consider the CDFG of a synchronous dataflow DSP application shown in Figure 3.2. Based on data dependencies, this application can be carried out in a minimum of three control steps (c_{steps}) using the CDFG-1 shown in Figure 3.3, with two adders and two multipliers. Such implementation provides a throughput of $1/3 = 0.33$ (for non-pipelined systems, throughput is the inverse of latency [27, 11], throughput modeling will be addressed later in this chapter). Another alternative consists of implementing the application with only one multiplier and two adders but in four control steps, as shown by CDFG-2 in Figure 3.3. In that case the throughput is 0.25. Based on the priority of throughput or area metric, the appropriate CDFG can be selected.

However, the inclusion of a reliability metric based on a fault recovery mechanism can make the case more complex and difficult to evaluate. When a resource fails (due to a configuration bit flip), an alternative schedule can be derived to continue the system operation using the remaining resources, most likely at a lower throughput. For example, to maximize the throughput, CDFG-1 is implemented. For a single component failure, e.g. a multiplier, the application can be rescheduled to implement CDFG-2 with lower throughput. For FPGA-based designs, such a fault recovery technique can be adopted as well. We will explore the dependability and performability-area tradeoffs for such systems. It is of interest that the controller for rescheduling the operations is assumed to be fault-free. This controller can be implemented in a separate chip, even in ASIC.

3.3 Methodology for design option analysis

In Figure 3.4, we present the detail of the *design option analysis* part of our proposed methodology that starts from the control dataflow graph of the application. As mentioned in chapter 1, the CDFG is extracted using the free academic tool known as GAUT [28]. It is worthy to mention that the boxes in the methodology represent steps, and the edges show the relationship between them. The steps are as follows:

1. *Configuration:* As we already know, A CDFG can be implemented with different component allocations (design options). To analyze each of these configurations, we model them separately with the PRISM modeling language. From now on, we will refer to the term *design options* as *configurations* in the rest of the chapter.

2. *PRISM modeling:* PRISM modeling requires the description of a system given in

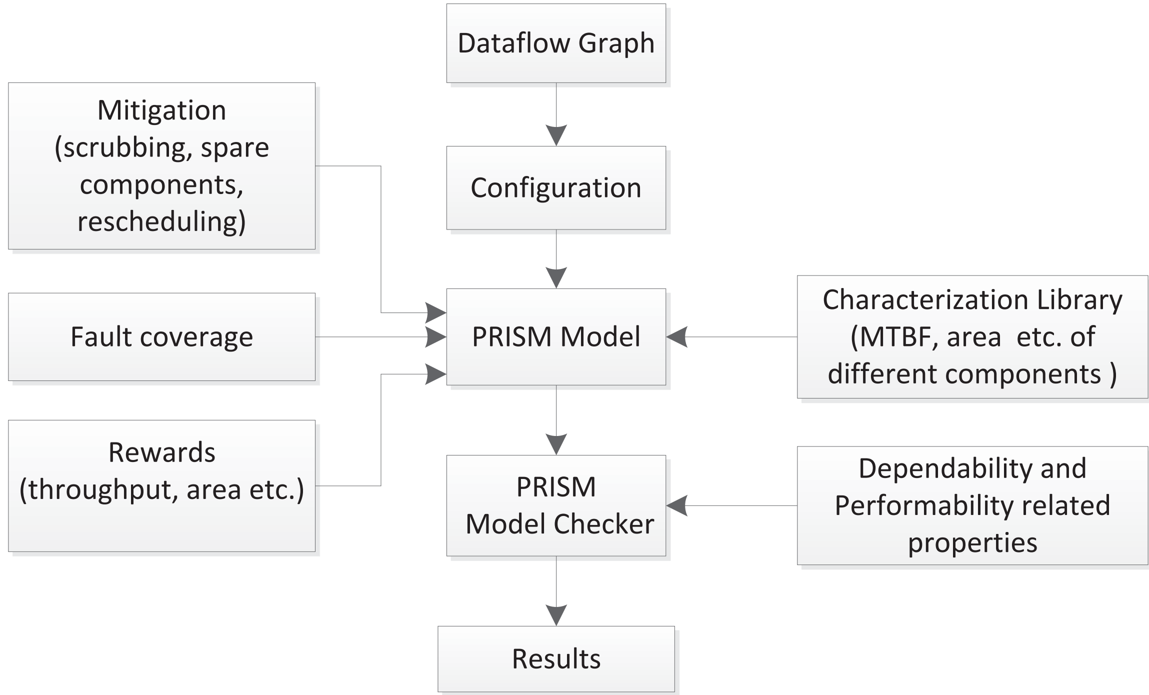

Figure 3.4: Design option analysis methodology

terms of component failure rates, adopted fault mitigation strategy, fault coverage and performance measures. To acquire the component failure rate, we use a *characterization library* (characterization library is explained in section 4.4). The modeled fault mitigation techniques are: rescheduling, cold spare components and blind scrubbing. For rescheduling a CDFG, if possible with available components, a high-level synthesis algorithm, such as *forced-directed list scheduling* [73] can be used. Since the model is parametric, the fault coverage value and the scrub interval can be varied for analysis. Each state of this Markov model can be augmented with associated rewards such as throughput (obtained using high-level synthesis techniques: CDFG scheduling with available components in each state), area (measured in terms of the total number of LUTs required to implement the design, obtained from component characterization library) or any other metric of interest. The resulting MRM is then analyzed using the PRISM model checker tool.

3. *PRISM model checker:* The PRISM tool then computes the set of all states which are reachable from the initial state and identifies any deadlock states (i.e. reachable states with no outgoing transitions). PRISM then parses one or more temporal logic properties (e.g. in CSL) and performs model checking, determining whether the model satisfies each property.

3.3.1 Markov Modeling of Reliability and Availability

CTMC models are very commonly used for modeling dependability of gracefully degradable systems. Each state in a CTMC model representing a specific configuration, can be classified into different types depending on the number of healthy components. For instance, the FIR filter in Figure 3.9 (quantitative results section) requires at a minimum an adder and a multiplier for successful operation. Hence, any state that does not fulfill the minimum resource availability is labeled as a *failed state*. At the end, the state labeled as *all fail* represents a state where all the components in the system have failed one-by-one due to SEUs. Note that *safe* and *unsafe* failures are not considered at this stage of modeling. How to include safety in the model will be described in detail in the next subsection. The initial state of a configuration has the maximum throughput and all the components are functional. The edges between the states represent transition rates. The assumptions for our model are defined as follows:

Assumption 1: The components fail independently and the time-to-failure for a component due to a configuration bit flip is exponentially distributed. Exponential distribution is commonly used to model the reliability of systems where the failure rate is constant. The *scrub* interval is assumed to follow an exponential distribution as well,

with a rate, $\mu = 1/\tau$, where τ represents the scrub interval.

Assumption 2: Every component in the system is connected with other components (via multiplexers). This assumption is needed for simplicity of the hardware model. The control unit can be designed as a finite state machine implemented either as a hardwired or microcoded controller. Since in many systems, datapath components dominate the area of the design compared to control units, these components can be much more vulnerable to SEU than control units. Hence, we only consider the failures of the datapath components in this work, and modeling of control units is left for future works.

Assumption 3: Only one component can fail at a time due to a SEU. This assumption is made to ensure the complexity in the Markov model is manageable. The system is assumed to be designed in such a way that each of its components can be easily diagnosed.

Assumption 4: *Cold spare* components are used to provide redundancy and are active only when a same type of component fails. The *cold spare* components are only error prone to cosmic radiations when they are active.

Assumption 5: The reconfiguration and rescheduling times (i.e. the time taken for the system to reschedule when a component fails and the time taken for repair via scrubbing) are extremely small compared to the times between failures and repairs. The time required for rescheduling is at most few clock cycles and the time required for scrubbing is only a few milliseconds.

Assumption 6: All the states in the CTMC model can be classified into three types: *operational*, where all the component are functional and the system has the highest throughput; *degraded*, where at least one of the components is faulty; and $failed$, where the number of remaining non-faulty components is not sufficient to perform successful operation and hence has a throughput of 0. In PRISM, a *formula* can be used to classify such states as shown in Figure 3.6.

Such a model is described as a number of modules in PRISM, each of which corresponds to a component of the system. Each module has a set of finite-ranged variables representing different types of resources. The domain of the variables represents the number of available components of a specific resource. The whole model is constructed as the parallel composition of these modules. The behaviour of an individual module is specified by a set of guarded commands. Once each module is specified in such a manner, the PRISM model checker then performs a parallel composition of all the modules to build the complete Markov chain of the system specified. A CTMC, as is the case here, can be represented in the following form in PRISM modeling language:

```
[] <guard> → <rate> :  <action> ;
```

The `guard` is a predicate over the variables of all the modules in the model. The update comprises of `rate` and `action`. A `rate` is an expression which evaluates to a positive real number. The term `action` describes a transition of the module in terms of how its variables should be updated. The interpretation of the command is that if the `guard` is satisfied, then the module can make the corresponding transition with that associated `rate`. A very simple command for a module with only one variable z

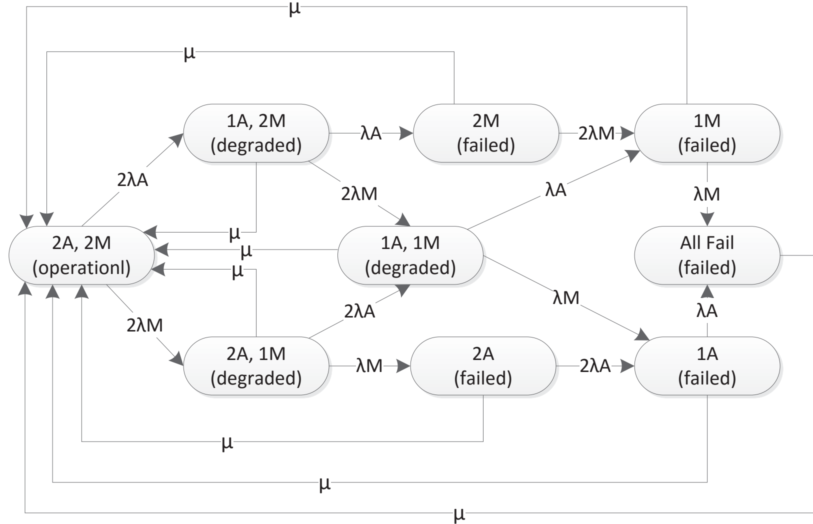

Figure 3.5: Sample CTMC for reliability/availability analysis

might be:

$$\texttt{[] <z = 0>} \rightarrow \texttt{7.5 : <z' = z + 1>};$$

which states that, if z is equal to 0, then it will be incremented by one and this action occurs with rate 7.5. A second more significant example, is an application that implements a function using 2 adders and 2 multipliers but that requires at least 1 adder and 1 multiplier (in case of failure due to SEU) for successful operation. Such a configuration in the PRISM modeling language can be described as shown in Figure 3.6.

In the PRISM code in Figure 3.6, `num_A` and `num_M` represent the number of adders and multipliers available in the initial state of the configuration. The `lambda_A` and the `lambda_M` variables represent the associated failure rates of the adders and multipliers whereas `miu` represents the repair rate. Each repair transition (scrub) leads back to the initial state reflecting the scenario that the configuration bit flips have been repaired. The value of `lambda_A` and `lambda_M` is obtained from a component characterization library, that will be explained later in this chapter. PRISM then constructs, from this,

```
module adder
 a : [0..num_A] init num_A;
 [] (a > 0) -> a*lambda_A : (a' = a - 1);
 [rep] (a < num_A) -> miu : (a' = num_A);
endmodule

module mult
 m : [0..num_M] init num_M;
 [] (m > 0) -> m*lambda_M : (m' = m - 1);
 [rep] (m < num_M) -> 1 : (m' = num_M);
endmodule

formula fail = (a =0)|(m =0);
formula oper = (a=num_A)&(m=num_M);
formula degrade = !fail & !oper;
```

Figure 3.6: PRISM modeling for a system with 2-adders and 2-multipliers

the corresponding probabilistic model, in this case a CTMC. The resulting CTMC for this configuration is shown in Figure 3.5 (`lambda_A`, `lambda_M` and `miu` are reflected in the figure as $\lambda A, \lambda M$ and μ respectively). The repair transition in the code is synchronized with a label `[rep]` to demonstrate the phenomenon that when the FPGA is scrubbed, all the components get fixed simultaneously. The formula `fail`, `oper` and `degrade` classifies *failed*, *operational* and *degraded* states in the model.

3.3.2 Safety Modeling using Fault Coverage

Any fault detection algorithm can be assumed to detect and handle all the faults properly. However, in reality this is not the case. A fault can escape the implemented fault detection mechanism. As a result, the system will not be able to reschedule, hence the system will be continuing its operation in a faulty mode. Which means, each component in the configuration that implements the CDFG, can fail either in a *safe* or in *unsafe* fashion. That is why we need to refine the model by taking into account and introducing the concept of the *safe failure* and *unsafe failure*. We define

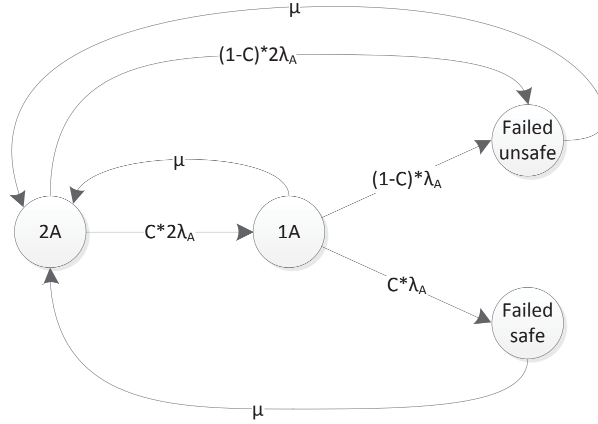

Figure 3.7: Safety modeling of simple system with safe and unsafe failure

them as follows:

Definition 1: *Safe failure* could be when a component's failure due to an SEU is properly detected, and handled by rescheduling depending on the number of remaining components. If the rescheduling is not possible (number of available components are less than the minimum number of components required for rescheduling), the system will move to a *fail safe* state.

Definition 2: An *Unsafe failure* is the fail silent behavior, observed when a system fails to detect a component's failure.

If all the faults are safely detected, it will eventually lead to the *failed safe* state, whereas even if there is a single *fail unsafe* occurrence, it will immediately lead to the *failed unsafe* state. The error detection coverage of a component can be defined by a conditional probability C :

$$C = P(fault\ detection | fault\ existence)$$

Figure 3.7 shows the modeling of safety for a simple single component system with only two adders including the repair transitions. For this case, we assume that the system requires at least one adder for successful add operation. Initially the system is in operational mode with two adders. When one adder fails, if the failure is detected, the system is rescheduled and continues with only one adder. If the failure is not detected, then it moves to the *failed unsafe* state. If another adder fails, the system will not be able to continue its operations, hence it will fail safely. However, if this failure is not detected, then the system will eventually fail in an unsafe fashion. Inclusion of safety in the model requires the modification of assumption 6 as follows :

Assumption 6: All the states in the CTMC model can be classified into four types:
1. *operational* - All the component are functional and the system has the highest throughput.
2. *degraded* - At least one of the components is faulty.
3. *failed safe* - The number of remaining non-faulty components is not sufficient to perform successful operation and hence has a throughput of 0. To reach *failed safe* state, all the failures leading to this state must be *fail safe.*
4. *failed unsafe* - At least one failure is not detected by the detection algorithm. *Fail silent* behavior of a component immediately leads to a *failed unsafe* state.

Figure 3.8 shows the modified PRISM code from Figure 3.6 after including the coverage variable c in the model.

3.3.3 Peformability Modeling using MRM

When a system changes its state from one to another one due to a full/partial failure or repair, the performance level can change. Such a scenario can be described by different

```
module adder
a : [0..num_A+1] init num_A;
[] (a  > 0 & (a < num_A+1)) -> c*a*lambda_A :
(a'=a-1) + a*(1-c)*lambda_A : (a'= num_A+1);
[rep] (a >= 0 )   -> repair : (a'=num_A);
endmodule

module mult
m : [0..num_M+1] init num_M;
[] (m  >  0 & (m < num_M+1)) -> c*m*lambda_M :
(m'=m-1) + m*(1-c)*lambda_M : (m'= num_M+1);
[rep] (m >= 0) -> 1 : (m'=num_M);
endmodule

formula fail_unsafe=((a=num_A+1)|(m=num_M+1));
formula fail_safe=((a=0)|(m=0))& !fail_unsafe;
formula oper =(a=num_A)&(m=num_M);
formula degrade=!fail_safe&!fail_unsafe&!oper;
```

Figure 3.8: PRISM modeling refined after inclusion of coverage *(c)* for a system with 2-adders and 2-multipliers

states using a Markov model that provides a framework for combined performance-reliability (performability) analysis. Formally, an MRM consists of a CTMC $X = X(t), t > 0$ with finite states space S, and a reward function r where $r : S \to \mathbb{R}$ [79]. For each state $i \in S, r_i$ denotes the reward obtained per unit time spent by X in that state which represents the the performance level given by the system while it is in that state.

Performability measures can be distinguished in different classes, mainly into two: steady-state performability and transient or point performability. For $i \in S$, let w_i denote the steady-state probability of residing in state i, and $p_i(t)$ the (transient) probability of residing in state i at time t. Steady-state performability (SP) can be defined as:

$$SP = \sum_{i \in S} w_i * r_i \tag{3.1}$$

Transient or point performability (PP) can be defined as:

$$PP(t) = \sum_{i \in S} p_i(t) * r_i \tag{3.2}$$

Markov Reward Modeling for the CDFG

For a data-flow system, the primary reward associated with each state of the MRM is throughput. For a synchronous data-flow system, the throughput can be evaluated directly from the CDFG of the system. As we consider only non-pipelined systems in our work, we can define the throughput as the inverse of the number of seconds it takes to execute the CDFG:

$$Throughput = (1/c_{step}) * (c_{step}/cycle) * (cycles/second) \tag{3.3}$$

where, c_{step} is the control steps in the CDFG. Assuming that each c_{step} takes a single clock cycle and η represents the system's clock frequency (clock cycle/second):

$$Throughput = (1/c_{step}) * \eta \tag{3.4}$$

In our MRM, the operational and degraded states are augmented with associated throughput reward, and all the *failed states* both safe and unsafe ones, are augmented with a throughput reward of zero. The expected throughput (for long run $E[X]$ or for a specific mission time $E[X(t)]$) can be calculated using the equation 3.1 and equation 3.2 respecively. In our MRM model, the area that is required, to implement the design on the FPGA, is assumed to be invariant between the states

for a specific configuration. The reason is, once the system is implemented on FPGA, the area is fixed (in terms of the total number of LUTs) and if a fault occurs, then the system will be rescheduled or if it fails, then eventually will be scrubbed. So only the control signals will change, not the components. For *overall reward* calculation e.g. to evaluate the throughput-area-reliability trade-offs for a configuration, we use the following equation:

$$Overall\ reward = (1/A) * E[X] \tag{3.5}$$

In the above equation, A represents the area of the design and $E[X]$ represents the expected throughput. This equation is similar to [52], however instead of calculating the reward up to a specified time-step, we use the notion of steady-state throughput. Such modeling can be considered as a direct optimization of throughput, area and reliability. Rewards can be weighted based on designer's requirements. For the case study presented in this chapter, the rewards are set to equal weight.

3.3.4 Characterization Library

As the SEU rate λ is highly dependent on device process technology, architecture, and orbits of interest, so this parameter is different for each device family. We use CREME96 [92] with radiation cross sections from [77] to find per bit upset rate λ_{bit} for Xilinx Virtex-5 in the Highly Elliptical Orbit (HEO) and Low Earth Orbit(LEO) orbit. The failure rate for a component can be calculated using the equation as follows:

$$\lambda_{component} = \lambda_{bit} \times Number\ of\ critical\ bits \tag{3.6}$$

For our experiments, $\lambda_{bit} = 7.31 \times 10^{-12}$ SEUs/bit/sec for the HEO orbit.

Table 3.1: Characterization library

Component	**No. of LUTs**	**No. of essential bits**	**MTBF (days)**
Wallace Tree Multiplier	722	133503	11.85
Booth Multiplier	650	130781	12.11
Brant-Kung adder	120	29675	53.36
Kogge-Stone Adder	183	41499	38.15

In order to build a component characterization library that represents the first-order estimation of the SEU effects on the components, we use the *bitgen* feature of Xilinx ISE tool. Using the *bitgen*, we identified the *essential bits* which is also known as *potentially critical bits*. Note that, it is well known that the number of *critical bits* is less than the number of *potentially critical bits*. More accurate SEU susceptibility analysis can be performed using the fault injection techniques [61, 50], however, for first-order worst-case estimation, it is valid to assume that all the *essential bits* are considered as *critical bits*. Note that we use the characterization library to obtain the failure rate of the components for the Markov chain model and the methodology is generic enough to be used with a different characterization library with more precise and accurate data, without any major changes.

Table 3.1 presents a first-order worst-case estimate of component failures due to SEUs. We characterize different adder and multiplier components, namely 64-bit Brent-kung adder, 64-bit Kogge-stone adder, 32-bit Wallace-tree multiplier and 32-bit Booth multiplier. The Xilinx Synthesis Technology (XST) tool is used to synthesize the components for Virtex-5 XC5VLX50T device from their HDL codes and the number of required LUTs to implement them (area) is also obtained. We observe that a 32-bit Wallace-tree multiplier has about 0.134 million bits that are sensitive to SEUs. So this multiplier has a worst-case Mean Time Between Failures (MTBF) of 11.85 days for space applications in the HEO orbit. MTBF and λ are

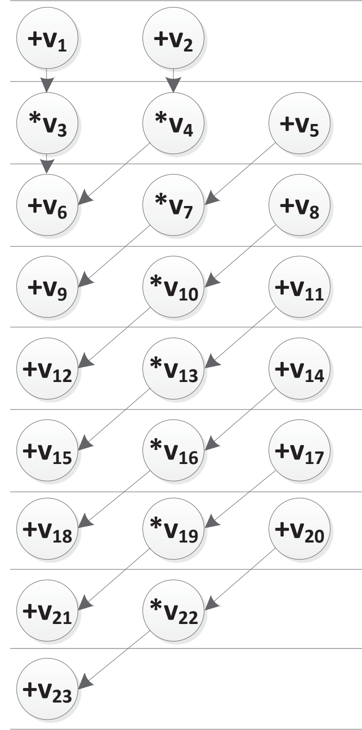

Figure 3.9: CDFG of an FIR filter

related to each other using the following equation [93]:

$$\lambda = \frac{1}{MTBF} \tag{3.7}$$

3.4 Quantitative Analysis using PRISM

Filters are commonly used in digital communication systems for different purposes, such as for equalization, signal separation, noise reduction and so on. Communication is a fundamental issue for any space-borne applications ranging from satellites to unmanned missions. That is why digital filters have an important role to play for such systems [72]. FIR filters are one of two primary types of digital filters (the other one is Infinite Impulse Response) used in Digital Signal Processing (DSP) applications.

Table 3.2: Available design options to evaluate

Configuration	Spare components	Scrubbing	Rescheduling
2A 2M	None	✓	✓
2A 3M	1 Mul	✓	✓
3A 2M	1 Add	✓	✓
3A 3M	1 Add, 1 Mul	✓	✓

FIR filters are commonly used in spacecrafts for noise filtering from images, videos and sensor outputs and spacecraft antennas [96, 34, 17]. To illustrate the applicability of the proposed methodology for early design decision, this section presents a Finite Impulse Response (FIR) filter case study [51] from the high-level synthesis benchmark [12].

Figure 3.9 shows the CDFG for a 16-point FIR Filter [45] obtained from [66]. To achieve a schedule with minimum number of control steps, the minimum allocation is two adders and two multipliers for the FIR filter application. At a minimum a pair of one adder and one multiplier is required for successful operation. For our experiments, we consider the 32-bit Kogge-stone adders and 32-bit Wallace tree multipliers as available components from the characterization library, as they require less area (number of LUTs) to implement. We must mention that any other adder or multiplier from component the characterization library can be used for the similar analysis. The first part of the case study presents the dependability analysis on different configurations. The latter part of the case study focuses on the performability (throughput with reliability) analysis and overall reward calculation. Overall reward (equation 3.5) gives the expected reward with both area and throughput taken into consideration.

Table 3.2 shows the different configurations to evaluate for FIR filter design and the their respective fault mitigation strategy. The first configuration consists of two adders and two multipliers with no redundancy. The second and third configuration consist of one spare multiplier and one spare adder respectively used as redundant

Table 3.3: Configurations vs classes of states

Config.	**I (days)**	**Operational (days)**	**Degraded (days)**	**Failure (days)**
C1	1	2989.00	609.04	51.94
	4	1937.53	1287.04	425.42
	9	1222.40	1378.28	1049.31
C2	1	2989.00	642.82	18.14
	4	1937.53	1492.61	219.86
	9	1222.40	1711.59	716.00
C3	1	2989.00	613.08	47.91
	4	1937.53	1319.58	392.88
	9	1222.40	1441.09	986.50
C4	1	2989.00	647.06	13.93
	4	1937.53	1531.90	180.55
	9	1222.40	1795.97	631.61

components (*coldspare*). Configuration 4 is equipped with full component-level redundancy, with a spare of each type of components. All the four configurations employ scrubbing and rescheduling. In rest of the chapter, configuration 1, 2, 3 and 4 will be referred as C1, C2, C3 and C4 respectively. Also for brevity when reporting experimental results, the scrub interval and fault coverage will be denoted by I (in days) and C respectively, and their units, when applicable, will be omitted.

In table 3.3, using reward-based properties, we analyze the number of days the design spends in different classes of states for a mission time of 10 years and fault coverage C = 0.99, with a value of I = 1, 4 and 9. Different states in the Markov model can be classified into various classes using *formulas* in PRISM language. To calculate the number of days spent in different classes of states, we define a reward structure for each of them. For example, a reward structure *degraded* assigns a state reward of 1 to all states of the model in which the system is in *degraded* mode. A property that can reason about the amount of rewards accumulated over a period of time, is represented using CSL logic in PRISM as follows:

Property 1: $R\{\text{"}degraded\text{"}\} =?[C <= t]$ - "the expected cumulative time spent in the degraded mode of the system in the time interval [0, t]".

The first column of the table shows the different configurations for evaluation and the second column shows the associated scrub intervals (I). The third, fourth, and fifth column presents the number of days the design spends in different classes of states. It is worth mentioning that the fifth column shows the days spent in either *failed safe* or *failed unsafe* states. All the configurations spend approximately similar number of days in *operational state* (rounded to 2 decimal points) for the same scrub intervals. For I = 9, configuration C1 that has no redundant components shows the worst result. Interestingly, we observe that adding an extra adder as spare does not help much whereas adding an extra multiplier as spare significantly reduces the number of days spent in *failed* states. In configuration C4, the added spares for both adder and multiplier provide the best result in terms of availability. This is obvious but will cost more area on the FPGA. Configuration C1 spends the least number of days and configuration C4 spends the highest number of days in *degraded* states. For many safety-critical applications, low performance for a period of time is acceptable. For such systems the number of days spent in *failed* states is a major concern and hence, configuration C4 and configuration C2 are the two best candidates.

Steady state analysis of a design is useful to evaluate its dependability in the long-run. In Figure 3.10, we calculate the steady-state failure probability (safe or unsafe) and compare the results of the four available configurations, with respect to different scrub intervals (I is varied from 1 to 7) and same coverage (C = 0.99). The steady-sate failure probability for a given configuration can be analyzed in PRISM

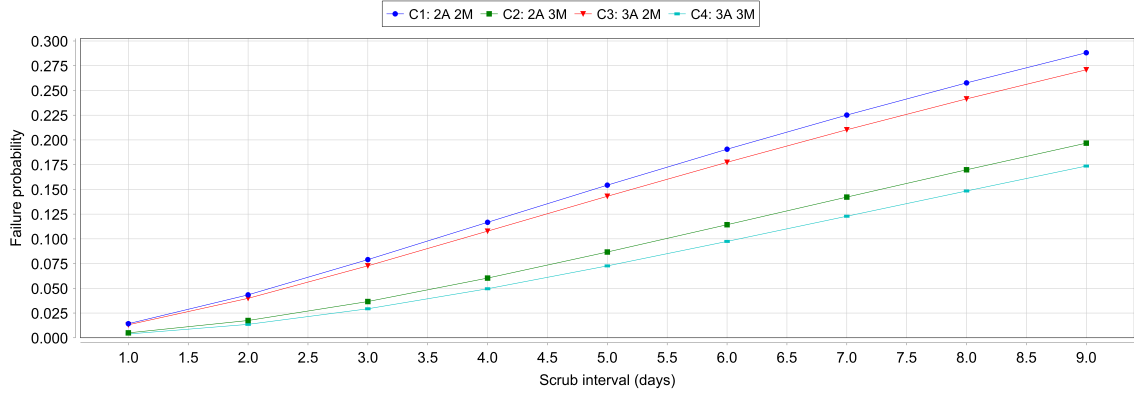

Figure 3.10: Failure probability vs I (scrub interval)

using the following property:

Property 2: $[S] = ?\ [\ failed_{safe} + failed_{unsafe}\ ]$ - "the long-run non-availability of the system".

The experimental results show that for configuration C1, the failure probability varies from 0.014 to 0.288 depending on the value of I. Configuration C2 has a lower failure probability than configuration C3 for all the scrub intervals. The failure probability of configuration C4 for all different scrub rates shows the best result with associated extra area overhead. From the results, we observe that configuration C2 is really an attractive alternative to configuration C4 (even for I = 7, the probability varies by only 0.023). On the other hand, configuration C1 and configuration C3 offer similar results over the long-run. Another conclusion that can be added is, for a value of $I > 2$, the failure probability increases sharply for all the configurations. For a value of $I \leq 2$, configuration C1 and configuration C3, and, configuration C2 and configuration C4 has almost same failure probability.

Figure 3.11 and Figure 3.12 show the effect of coverage C on reliability and

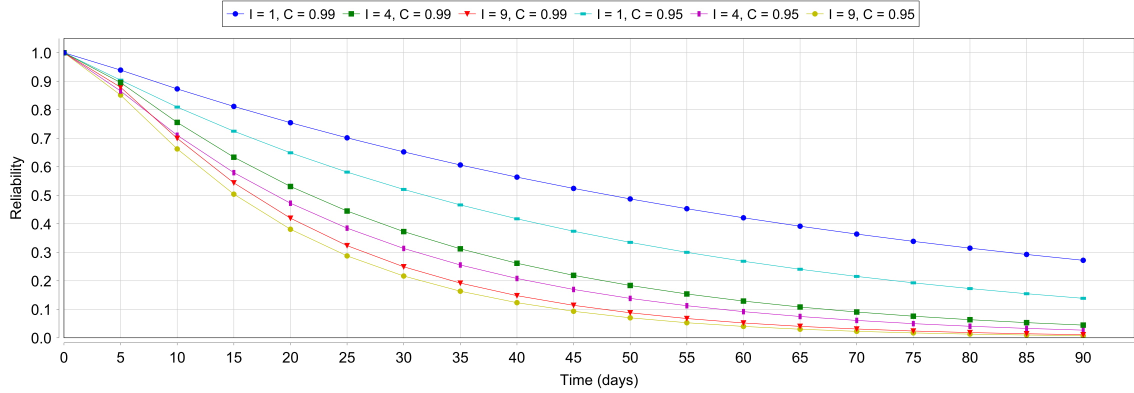

Figure 3.11: Reliability vs I (scrub interval)

safety respectively, for different values of I, for a mission time of maximum 3 months (T is varied from 1 to 90 days). For this part of the experiment (reliability and safety analysis), we consider configuration C1 with two adders and two multipliers, however any other configuration can also be analyzed in the similar fashion. The properties used to analyze reliability and safety in PRISM are as follows:

Property 3 (Safety) : $P = ?\ [\ G\ [0,\ T]\ operational \mid degraded \mid failed_{safe}\ ]$ - "The probability that the system will be either in *operational*, *degraded* or in *failed safe* state in first T days".

Property 4 (Reliability) : $P = ?\ [\ G\ [0,\ T]\ operational \mid degraded\ ]$ - "The probability that the system will be either in *operational* or in *degraded* state in first T days".

Figure 3.11 shows some interesting results for reliability evaluation. Configuration C1 has the highest reliability for I = 1 and C = 0.99. We observe that, with the same coverage, for a delayed scrub of I = 4, configuration C1 has lower reliability than the

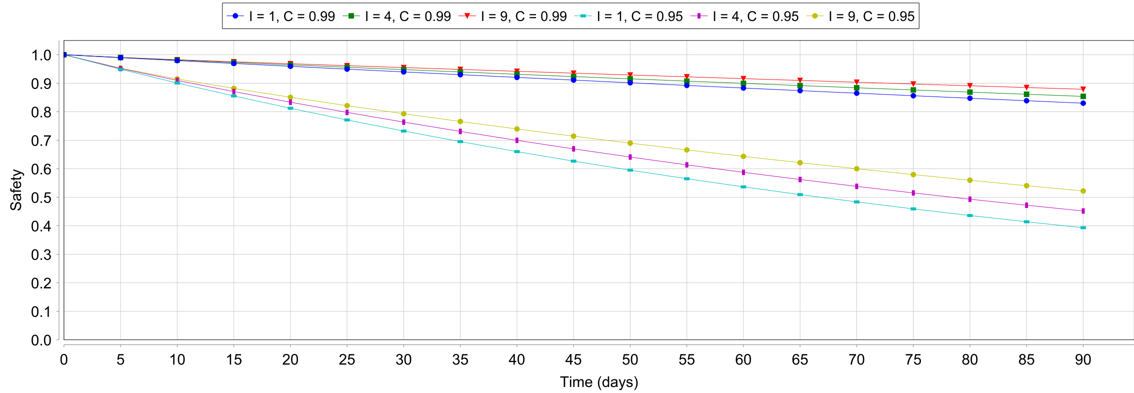

Figure 3.12: Safety vs scrub I (scrub interval)

same configuration with I = 1 and C = 0.95. So, a high coverage does not by itself guarantee a high reliability, particularly if the scrub interval is long. In contrast, if the scrub interval is fixed, then increasing the coverage will always increase the reliability. For example, the design with I = 4 and C = 0.99 has a higher reliability compared with the design with I = 4 and C = 0.95. In Figure 3.12, we observe that, for C=0.99, the safety of the system never goes below 0.83 in the first 3 months, even for the most delayed scrub interval (I = 9). When we analyze the system for C = 0.95, it shows how drastically the safety of the system falls. For C = 0.95, the safety of the system drops up to 0.39 for a mission time of 3 months. It is also noticeable that if the value of I increases, then the distance between the safety values also get wider even for the same coverage.

Figure 3.13 reveals an important observation to compare the available design options. We compare configuration C1 with no redundancy and configuration C4 with full redundancy for three different values of coverage C and I = 1 (since the model is parametric, any other parameter combinations can also be easily evaluated). We observe that, for perfect coverage (C = 1), indeed the configuration with redundancy

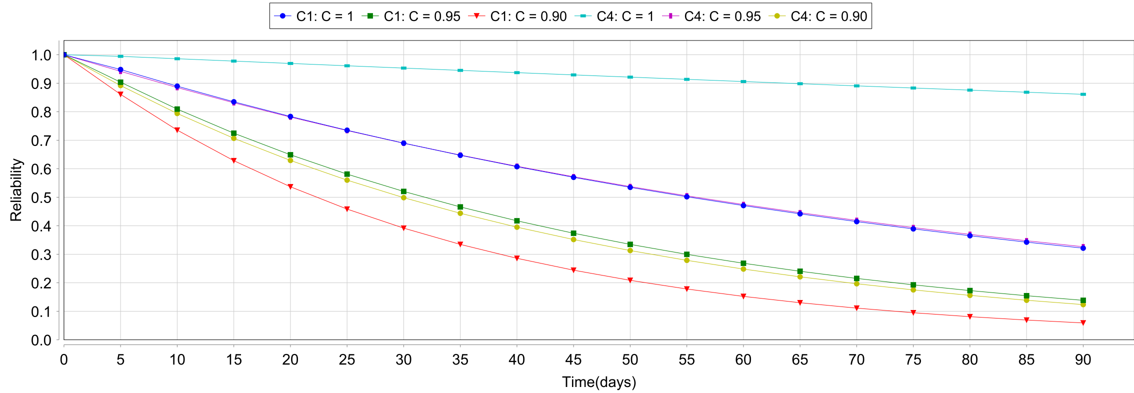

Figure 3.13: Impact of C (coverage) on the design with/without redundancy for I = 1)

gives better reliability. However, for lower coverage values, such as C = 0.95, configuration C4 with redundancy gives almost the same reliability compared to the configuration C1 with no redundancy with perfect coverage. For even lower coverage value, redundancy fails to improve the reliability compared to the configuration C1 for the cases where it has better coverage. This experiment shows that a design option with redundancy is not always the best choice with lower coverage. For instance, all C4 curves for which $C < 0.95$ produce a reliability less than C1 with C = 1. We redo this experiment for a delayed scrub, I = 9 and plot the results in Figure 3.14. This experiment shows that if a design option has coverage more that 0.85, then the design option with redundancy provides a better reliability. From this, we can conclude that, if a system employs longer scrub interval, then a design option with redundancy can provide better reliability even with lower coverage, compared with the design option with no redundancy, with same coverage. However, if the coverage goes lower beyond a certain point, indeed redundancy will not help improving the reliability. Comparison of Figure 3.13 and Figure 3.14 also indicates that redundancy is more useful for improving reliability in the cases where scrub interval is longer. For systems with fast scrubbing capability, rescheduling can be a good alternative to redundancy-based

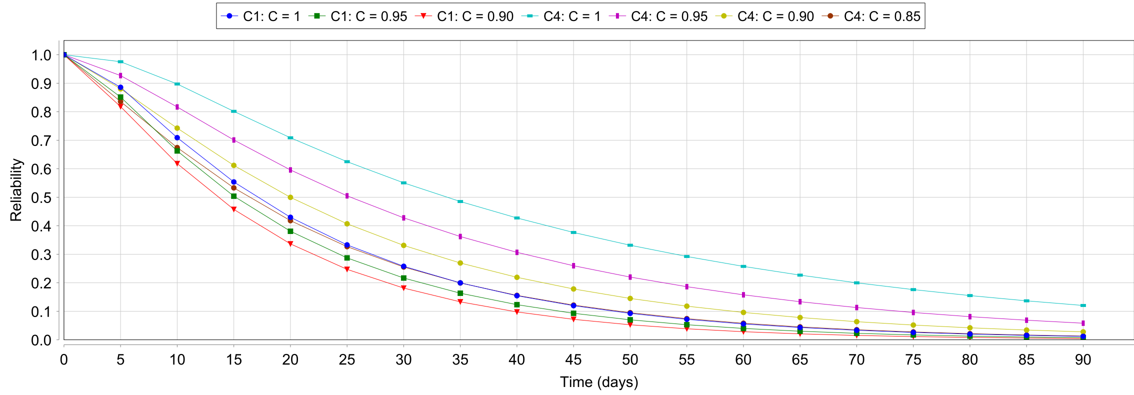

Figure 3.14: Impact of C (coverage) on the design with/without redundancy for I = 9)

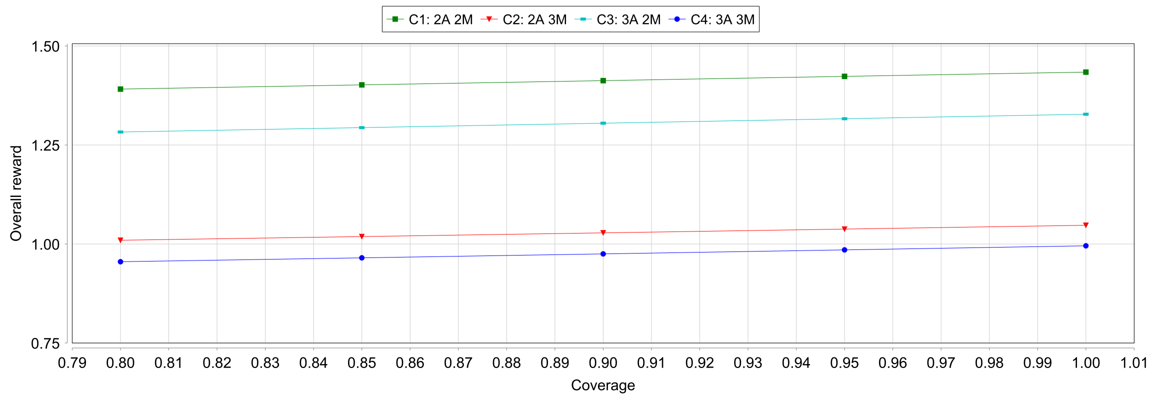

Figure 3.15: Impact of C (coverage) on for performability-area trade-off evaluation for I = 1

solutions.

For performability and throughput-area analysis, Table 3.4 shows the expected throughput and long-run *overall reward* calculation for various scrub intervals with C = 0.99. The rewards are setup so that the area and expected throughput have equal weights. Both the area and throughput were normalized between 0 and 1 in order to not skew the reward numbers. For every configuration, the maximum throughput (throughput in the initial state) is used to normalize the throughput for other states in the Markov reward model. Similarly, the maximum area is used to normalize the other area values among different configurations. In our model, a reward structure

Table 3.4: Overall reward calculation

I (days)	Config.	Normalized Expected Throughput	Area (No. of LUTs)	Norm. Area	Overall Reward
1	C1	0.955	1810	0.667	1.432
	C2	0.974	2532	0.932	1.045
	C3	0.973	1934	0.734	1.326
	C4	0.993	2765	1.000	0.993
4	C1	0.811	1810	0.667	1.216
	C2	0.876	2532	0.932	0.940
	C3	0.856	1934	0.734	1.166
	C4	0.931	2765	1.000	0.931
9	C1	0.628	1810	0.667	0.942
	C2	0.717	2532	0.932	0.769
	C3	0.684	1934	0.734	0.932
	C4	0.790	2765	1.000	0.790

throughput assigns a normalized throughput reward to all the operational or degraded states. All the *failed safe* and *failed unsafe* states are augmented with a throughput reward of zero. Steady-state expected throughput (normalized) for a configuration can be analyzed in PRISM using the property as follows and shown in column 3:

Property 5: R {"*Expected throughput*"} = ? [*S*] - "The expected throughput of the system".

Column 4 shows the area of each configuration and their normalized value is shown in column 5. Column 6 shows the overall area-throughput reward (overall reward) for each configuration. The reward for each configuration is calculated by multiplying the value of column 2 with the reciprocal of the normalized area. Based on the equal reward weighting, configuration C1 which has no redundancy (spare components), shows the best throughput-area reward for all the values of I. This indicates that the extra reliability provided by the redundancy is not always useful to suppress the extra

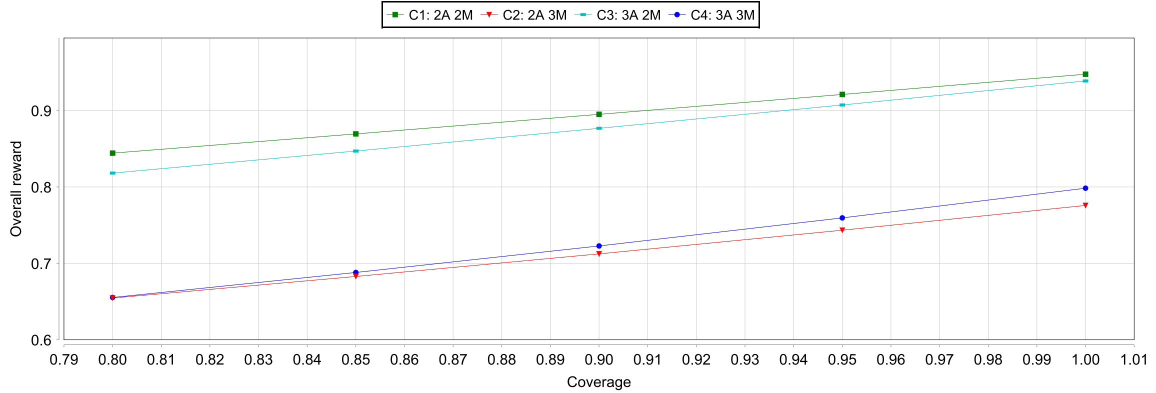

Figure 3.16: Impact of C (coverage) on for performability-area trade-off evaluation for I = 9

area overhead. However, rescheduling with scrubbing is good enough to serve as a fault recovery and repair mechanism in such cases. Another important observation is that adding a spare adder significantly improves the throughput-area reward, much more than adding a spare multiplier. If performance is the main concern of the design, then the expected throughput results from column 3 suggests configuration C4 as the best choice to implement. It clearly shows, how the inclusion of throughput-area metrics can influence design decisions toward solutions that differs from those resulting from an analysis based on either dependability (as in Table 3.3) or performability metric alone. Such an analysis, using the proposed methodology, can be very useful at early design stages for designers of safety-critical applications concerned with dependability, performance and area constraints.

To analyze the impact of coverage on performability-area trade-off, we evaluate property 5 for scrub interval I = 1 and show the result in Figure 3.15. We find that, from lower to higher coverage, the trend is the same, configuration C1 with no redundancy keep dominating the overall reward graph. This supports the conclusion derived from Table 3.4 that redundancy is not always useful to suppress the extra area overhead for all coverage points. In contrast, when we redo this experiment for

a comparatively delayed scrub I = 9, we clearly notice the relationship between I and C reflected on the overall reward as shown in Figure 3.16. The configuration C1 with no redundancy is still dominating, but the rewards accumulated by configuration C3 approaches closer to configuration C1 with increasing values of C. In contrast, for lower coverage values, configuration C2 and configuration C4 accumulate almost similar reward, however the gap between them expands with increasing values of C. Such phenomena was not observed in Figure 3.15, but in Figure 3.16 it is visible for delayed scrub interval.

3.5 Summary

In this chapter, we illustrated how available design options can be modeled using a Markov (reward) model that captures the possible failures, fault detection coverage and repairs possible in radiation environment. Afterwards, a wide range of properties are exhaustively and automatically verified to evaluate the design options, in terms of throughput, area and dependability. Such analysis is useful to reduce the overall design cost and effort. The quantitative results from our obtained model shows some important observations such as the fact that high coverage is not always helpful for gaining high reliability, and that scrubbing delay has also a considerable impact. In terms of safety, we also showed how scrubbing interval affects safety of available design options with the same fault coverage. Our analysis also shows that redundancy may fail to improve reliability if it has lower coverage compared to a design with no redundancy but high coverage for some cases. For performability-area trade-off analysis, we showed that redundancy-based solutions might not be always the best choice as one may expect. Alternatively, for those cases, *rescheduling* in conjunction with *scrubbing* can be a good option. To our knowledge, this is the first attempt to

evaluate such relationships at early design stages using probabilistic model checking.

In the next chapter, we will explore how to model discrete time delays using Erlang processes for modeling scrub intervals with better accuracy. At the same time, we will also show in that chapter how the Design Assurance Level (DAL) compliance can be checked at system level using our methodology.

Chapter 4

Scrub Optimization and DAL Analysis

The number and the complexity of electric components in commercial aircrafts have grown dramatically. As a result, it became essential for the Federal Aviation Administration (FAA) to establish a baseline of required design flow steps for an airborne component. In 2005, DO-254 [65] was formally recognized as a standard to ensure the highest level of safety in airborne electronic systems. It includes five levels of compliance, commonly known as Design Assurance Levels (DALs). DALs range in severity from A (means that a hardware failure would cause a catastrophic failure of an aircraft) to E (means that a failure would not affect the safety). As expected from the description, meeting a DAL-A level of compliance requires significantly more effort and also greater attention to verification than would DAL-E. Figure 4.1 shows the DALs as mentioned in DO-254.

For applications implemented in SRAM-based FPGAs, the most traditional way of handing SEUs is to use TMR with scrubbing. We already know that implementing a TMR in a design increases the power consumption by 300%. Also, we know that

Level	Classification	Failure Condition Description	Pb/h
A	Catastrophic	Failure conditions that would prevent continued safe flight and landing.	$<10^{-9}$ Extremly improbable
B	Hazardous / Severe-Major	Large reduction in safety margins or functional capabilities, physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely, or adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants	$<10^{-7}$ Extremly remote
C	Major	Significant reduction in safety margins or functional capabilities, a significant increase in flight crew workload or in conditions impairing flight crew efficiency, or discomfort to occupants, possibly including injuries.	$<10^{-5}$ remote
D	Minor	Slight reduction in safety margins or functional capabilities, a slight increase in flight crew workload, such as routine flight plan changes, or some inconvenience to occupants	$<10^{-3}$ Probable
E	No Effect	Failure conditions that do not affect the operational capability of the aircraft or increase flight crew workload.	-

Figure 4.1: Design Assurance Levels

scrubbing a design less frequently increases the chances of accumulating SEUs in the design that will eventually break the TMR. On the other hand, frequent scrubbing consumes high power [67]. This indicates that a design should be scrubbed according to its dependability requirements, not too frequently, specifically if the design has strong power constraints like deep space missions.

In this chapter, we will explain the *scrub optimization/DO-254 Analysis* part of our methodology. This part of the methodology focuses on the use of probabilistic model checking technique for two purposes: (1) to analyze a reconfigurable system to validate if the design meets the assurance level (DAL) and also the high-availability requirements. (2) to explore the effects of the scrub interval on the design (modeled using Erlang process) to suggest the lowest possible scrub frequency to meet the availability requirements and also to assess reliability enhancements that can be obtained with TMR.

4.1 Erlang Distribution

A system exposed to a harsh radiation environment will eventually fail, requiring repair or replacement. Hence, it is important to have a model that can represent maintenance effects on a system's condition as well as the deterioration process. However, many systems subject to maintenance and degradation may involve state transitions, which depend explicitly on time, or occur discretely. For these reasons, maintenance actions cannot generally be modeled by a simple exponential distribution within the Markov process. For example, a significant repair or periodic inspection time, which may reflect realistic maintenance activities, does not generally follow the exponential distribution. Therefore, we have to develop an approximation methodology to allow the Markov processes to model significant holding times. In our modeling, the deterministic repair intervals are modeled using the Erlang process [30] for better accuracy. Using Erlang process a nonexponential holding time distributions can be approximated by inserting multiple intermediate states between every main state pairs.

Erlang distribution is (a special case of a phase-type distribution [70]) the most suitable phase approximation for the deterministic distributions since this is the least variable phase type distribution for any given number of phases [30]. A random variable X has an Erlang-k (k is the number of phases, k = 1, 2, ...) distribution with mean k/μ if X is the sum of k independent random variables $X_1, ..., X_k$ having a common exponential distribution with mean $1/\mu$ (μ is the scale parameter, also can be written as the inverse of rate parameter $1/\lambda$). The probability density function f(t) of Erlang is given by:

$$f(t) = \mu \frac{(\mu t^{k-1})}{(k-1)!} e^{-\mu t} \tag{4.1}$$

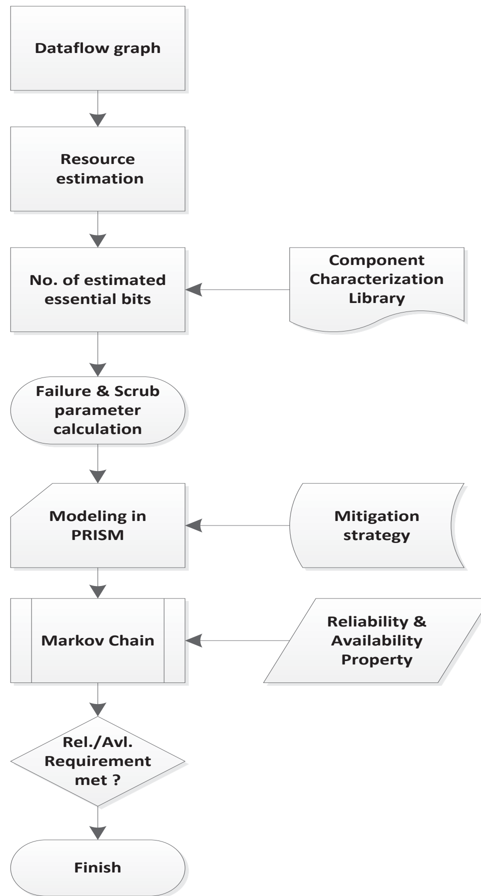

Figure 4.2: Scrub optimization/DAL analysis methodology

PRISM model checker is based on the use of classical Markov chain. Hence, the transition delay between a pair of states is exponentially distributed. Since configuration scrubbing is executed periodically after a fixed interval, to model this phenomenon approximation of discrete time delay is required. In this chapter while discussing the modeling, we will demonstrate how we use Erlang distribution in our modeling to achieve this goal.

4.2 Scrub optimization/DAL Analysis Methodology

In Figure 4.2, we present the methodology for scrub optimization and DAL analysis. We start from the dataflow graph of the application. Once the CDFG is extracted, a resource estimation process is used to estimate the resource required for the application. Resources estimation is based on the extracted CDFG. For this step, we analyze each of the nodes in the graph to compute resources required to implement an application on a specific FPGA target platform. A PRISM model is then built to analyze the design and this model is configured using environmental, target system and mitigation parameters. Different reliability and availability properties are then verified to check if the design meets the reliability and availability requirements. The different steps of the methodology are explained as follows.

4.2.1 Resource Estimation

The resource estimation procedure is inspired from [90], and based upon the concept of primitives, which represent elementary functions (e.g. addition, multiplication) used in target applications. The primitives can be divided in two types: 1) the functional primitives, which are the basic operations as they appear in the C/C++ code and the CDFG, and 2) the structural primitives, which are the hardware counterparts of the functional primitives appearing in the target FPGA. Based on these, the estimation procedure simply becomes identifying functional primitives in the CDFG and matching them with structural primitives, which appear in the target characterization library.

Estimated resources also depend on the style of application of the design. For

Figure 4.3: A simple Markov chain to illustrate failure occurrence

example, a full parallel application of a CDFG will require maximum number of resources with maximum speedup. However, depending on area, performance and power requirement, the CDFG might require scheduling to deal with the constraints. Depending on the required resources, using a characterization library (explained in the previous chapter), the number of essential bits is estimated for the design. The number of essential (potentially critical) bits is important for calculating the Markov model parameters.

4.2.2 Markov Modeling of Failure and Deterministic Delay

It is known that the probability of a state transition for a classic Markov model is assumed to depend only on the current state. This is equivalent to assuming that failure rates are constant and that failure occurrence is a Poisson process. Since SEU rates are usually modeled using a Poisson process [18], that implies that the time between two consecutive events is exponentially distributed.

```
module adder
 s : [0..2] init 0;
 [] (s = 0) -> lambda_1 : (s' = s + 1);
 [] (s = 1) -> lambda_2 : (s' = s + 1);
endmodule
```

Sample PRISM code

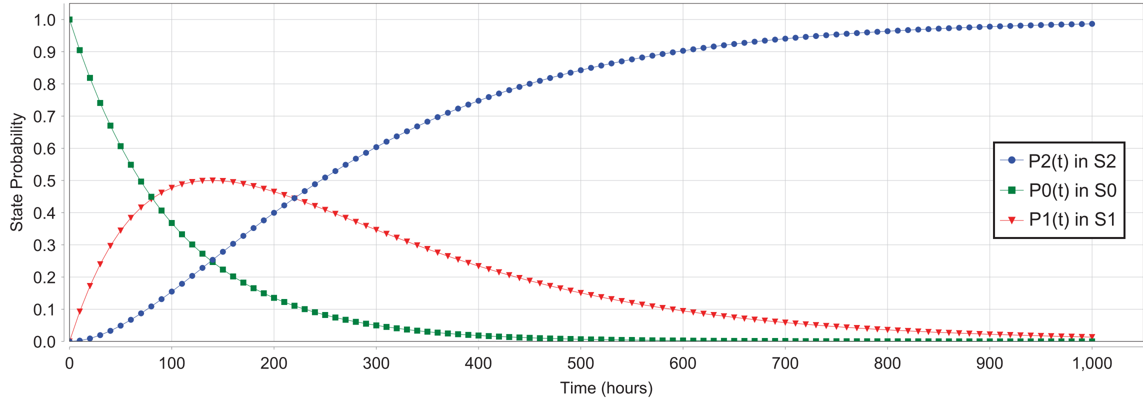

Figure 4.4: State probabilities vs time

Lets assume that we have a system with error detection capability. The Markov model of such a system as shown in Figure 4.3, can be built with three discrete states (S0: fully operational, S1: faulty but with fault undetected, and S2: fault detected, failed) representing the system's status. However, the number of states can be easily changed, depending on the degree of model specificity. The failure rates λ_1 and λ_2 are constant between states. This system can be described using PRISM modeling language as shown in the sample PRISM code.

For the sample Markov chain in Figure 4.3, if $\lambda_1 = 0.010$ and $\lambda_2 = 0.005$, then the corresponding state probabilities, and reliability function of that Markov degradation model can be generated using PRISM as displayed in Figure 4.4 and Figure 4.5. The reliability function is calculated by summing up all the state probabilities except that of the failed state, S2. In this example, the system is considered tolerant to only one fault. Conversely, a Markov process can also be derived to approximate a given reliability function.

As mentioned earlier, we use the concept of a phase-type distribution to approximate a time delay until absorption to one of the states in the Markov chain. It is also known that the Erlang process (i.e., summation of identical exponential distributions

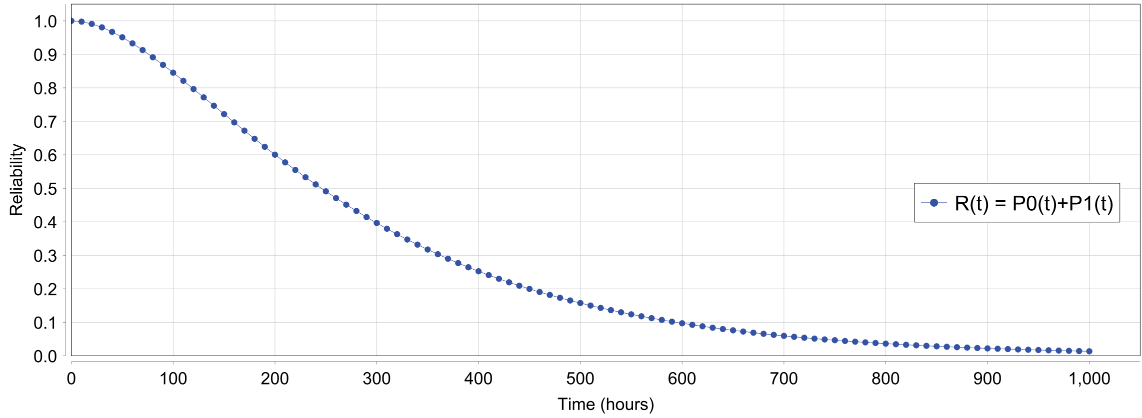

Figure 4.5: Reliability vs time

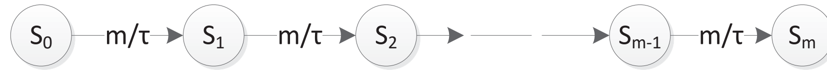

Figure 4.6: Markov chain for Erlang process

as displayed in Figure 4.6) minimizes the variance among any phase-type distributions [30]. In other words, non-exponential holding time distributions can be approximated by inserting multiple intermediate states between the two conventional degradation states. For illustration, in Figure 4.6, τ is the total transition interval between S_0 and S_m, and $m-1$ is the number of intermediate stages used to approximate it. The rate at which transitions happen is proportional to m to provide a same total transition time. This Erlang process approximation of a constant time delay in a Markov process enables the incorporation of various maintenance activities into the equipment deterioration model.

Figure 4.7 illustrates the results of implementing the Markov chain of Figure 4.6 to approximate a constant delay of 10 hours. The figure shows the probability distribution of delay times for different values of m. This is how we implement constant time delay for modeling of periodic repair while preserving the Markov property. There is a clear and obvious trade-off here between the accuracy (how close it is to modelling

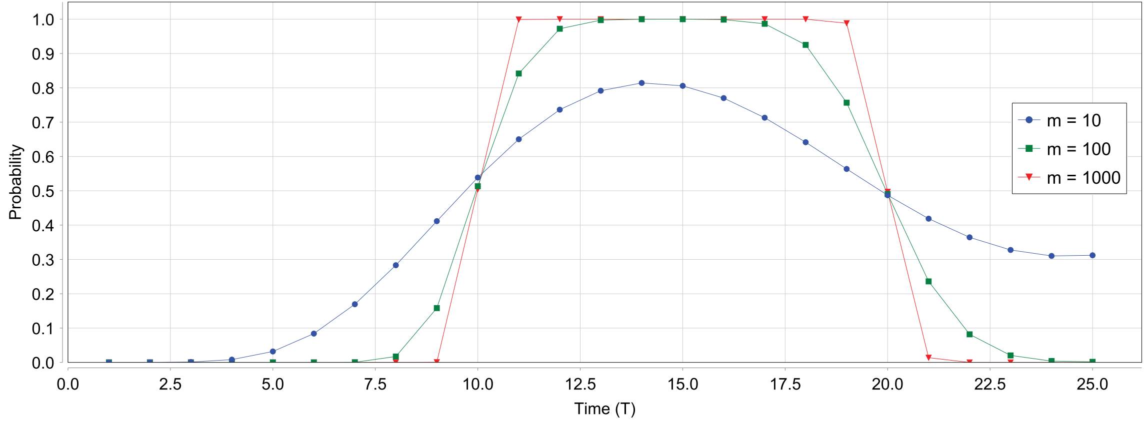

Figure 4.7: Deterministic delay modeling with Erlang process

a deterministic time delay) and the resulting expansion in the size of the model.

4.2.3 Modeling Scrub and TMR

A system that utilizes the periodic blind scrub mitigation technique can be modeled as a Markov model as illustrated in Figure 4.8. Here the states represent:

S_{0i} : The system is fully operational $(1 < i < m)$;

S_{1i} : The system is faulty with one or more faults $(1 < i < m)$;

In this model, λ_{design} represents the failure rate of the design and μ represent the repair rate, where μ = m/τ , τ = scrub interval. The Markov process is created by stacking the Erlang processes (shown in Figure 4.6) on top of the failure model (with 2 main states) shown in Figure 4.3. The conceptual block diagram of TMR using device-level redundancy is shown in Figure 4.9. Three implementations of the design (known as Functional units or FUs) are used in parallel, and their output goes to a majority voter circuit (the majority voter circuit is assumed to be fault free). The output with the majority vote goes to the final output. If more than two FUs

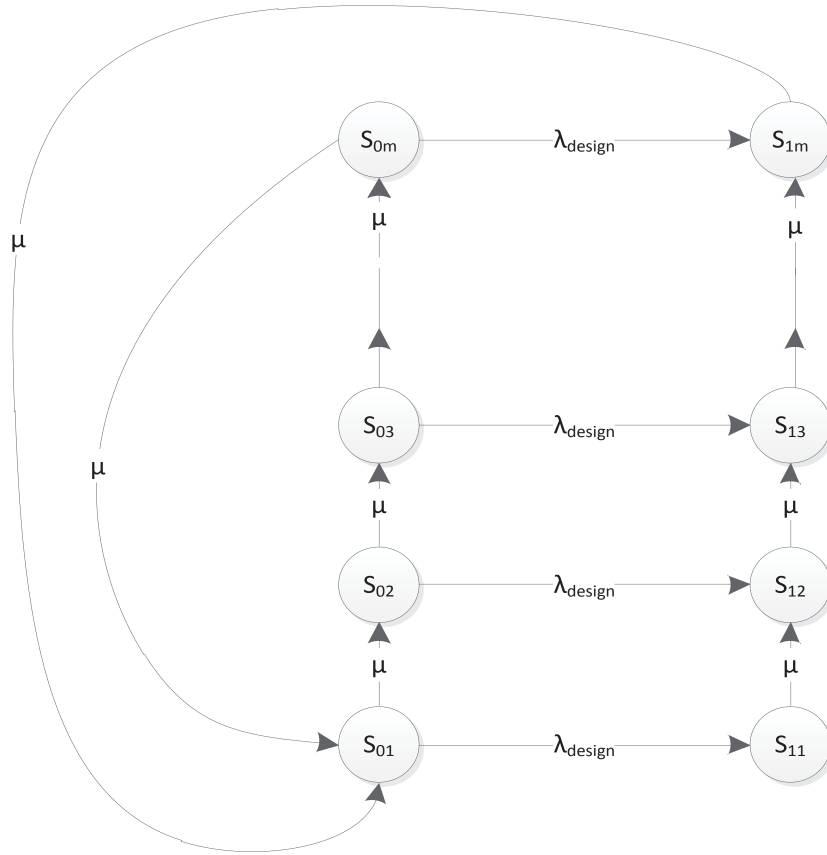

Figure 4.8: Markov model of periodic blind scrub using Erlang process

encounter errors, then the TMR strategy may fail, since the erroneous output can propagate. The Markov model of a system implementing both TMR and scrub is shown in Figure 4.10. This is an extension of the previous scrub only model. The states represent:

S_{0i} : The system is fully operational, e.g. All 3 FUs are fault free. $(1 < i < m)$;

S_{1i} : One of the FUs has encountered at least one SEU and thus the system is under fault, but the output is not erroneous. The system is in a degraded mode $(1 < i < m)$;

S_{2i} : Two of the FUs are faulty, caused by one or more SEUs in each FU, and hence the output may be erroneous. The system has entered a possibly failed mode and will

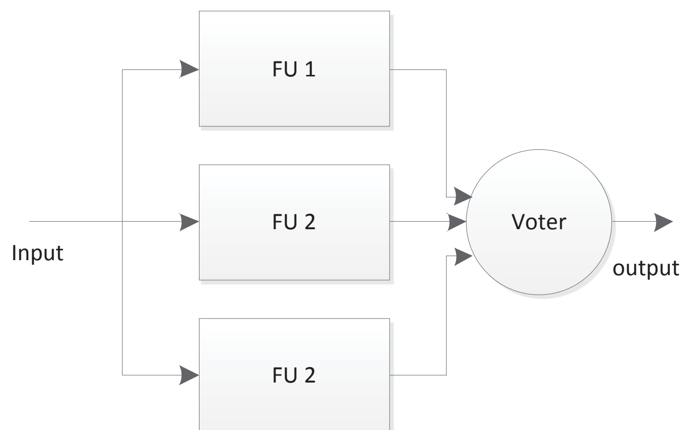

Figure 4.9: Conceptual TMR model

stay there until the next scrub arrives ($1 < i < m$);

4.2.4 Markov Model Parameters

Once the Markov model is built, we need to populate the model for further analysis. Three different types of parameters are required [62], namely (1) Environmental parameters, (2) Target system parameters and (3) Mitigation parameters. Some of these parameters, such as mitigation parameters, can be varied freely to achieve optimal availability. On the other hand, target system parameters and environmental parameters are fixed for a given target system and environment.

Environmental parameters

The key environmental parameter is the upset rate λ experienced in various orbits of interest. As the observed upset rates are dependent on device process technology and architecture, so this parameter may be different for each device family. We use CREME96 [92] with radiation cross sections from [77] to find per-bit upset rate λ_{bit}

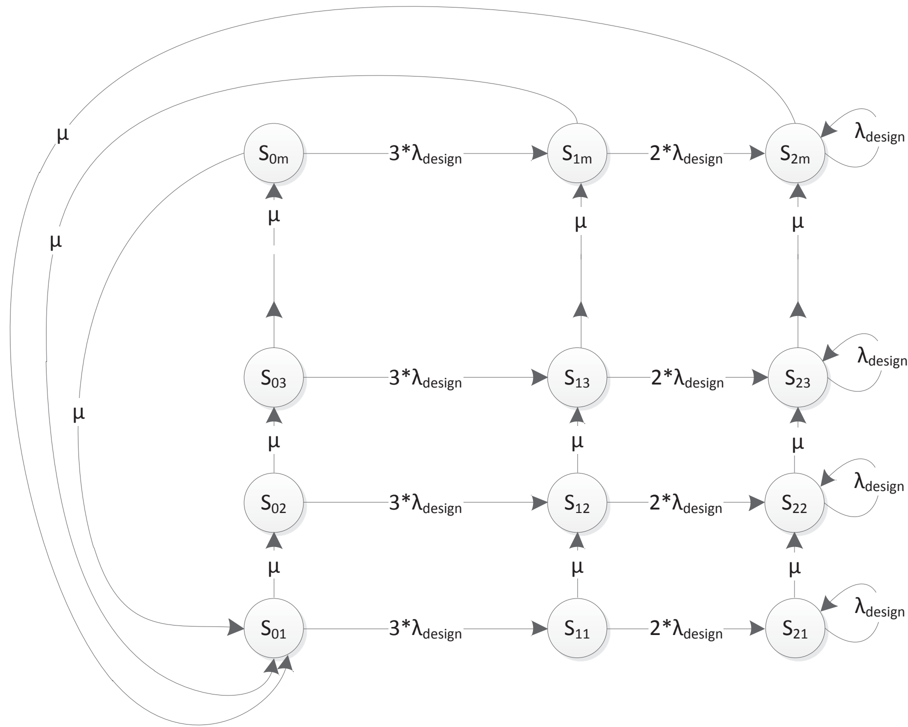

Figure 4.10: Markov model of TMR with periodic blind scrub using the Erlang process

for Xilinx Vitex-5 in ISS LEO orbit, which is 2.63×10^{-12} SEUs/bit/sec. The failure rate for this system can be calculated as follows:

$$\lambda_{design} = \lambda_{bit} \times Number\ of\ critical\ bits \tag{4.2}$$

The *number of critical bits* is the summation of the critical bits from the required resources estimated by the resource estimation process.

Target system parameters

The target system can be defined with three main parameters, namely SelectMAP bus width (B) and the configuration clock frequency (f_{clk}). Usually, these parameters are set by the system designer and also limited by the system architecture. They directly impact system availability. We assume a conservative system using a

Table 4.1: Model construction time and statistics

m	No. of states		No. of transitions		Time (s)	
	Scrub only	Scrub & TMR	Scrub only	Scrub & TMR	Scrub only	Scrub & TMR
50	100	150	150	300	0.006	0.006
100	200	300	300	600	0.008	0.009
200	400	600	600	1200	0.015	0.018

radiation-hardened memory with an 8-bit bus at 33MHz.

Mitigation parameters

For a system using periodic scrub, mitigation parameters μ_{design} describes the scrub rate of the system and it can be calculated using the target system parameters. Scrub rate can be determined experimentally, however this rate also can be estimated analytically by using the following equation:

$$\mu_{design} = \frac{B \times f_{clk}}{Total\ configuration\ bits} \tag{4.3}$$

We use this equation to calculate the repair rate parameter μ in our model, where $\mu = m \times \mu_{design}$.

For the purpose of parameter calculation in our case study, we consider the Xilinx Virtex-5 XC5VLX330 device as the target which has 79,704,832 configuration bits and ISS LEO orbit as the target orbit. Similarly, any other target device or any other orbit can be evaluated using the same methodology.

4.3 Case Study

For the case study, we consider a 512-tap parallel FIR filter for space application. Using the methodology described in [90], from high-level design description, the data

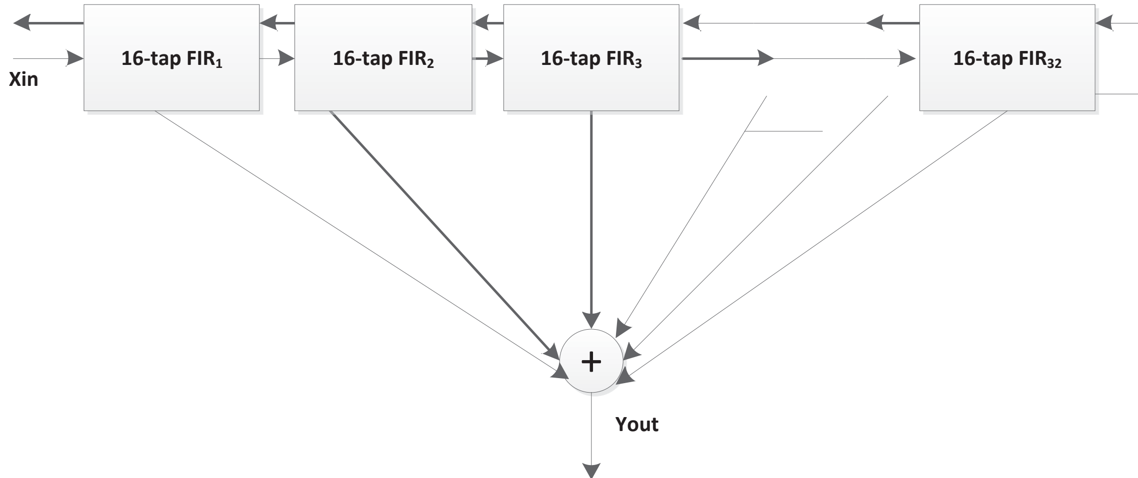

Figure 4.11: 512-tap parallel FIR filter

flow graph representation is obtained and the number of essential bits for FPGA implementation is estimated. In our experiments, we consider a worst-case scenario, where all the essential bits are considered critical bits. Figure 4.11 shows the block diagram of a 512-tap parallel FIR filter using 32 blocks, each of which embedding a 16-tap FIR filter. The number of essential bits required to implement this design is approximately 5863557 bits. Table 4.1 shows the model generation statistics and timings for different values of m (number of Erlang steps). For our experiments, we choose the value: $m = 200$.

The experimental results can be divided into two different sections: Analysis and Verification. In the analysis section, we show the use of probabilistic model checking to analyze the system and to evaluate its reliability and availability properties. In the verification section, we show how such analysis can be used to verify that the system meets its DAL and availability requirements.

4.3.1 Analysis

The scrub parameter μ is the reciprocal to the time needed to repair an upset once the scrub technique is triggered. From mitigation parameter calculation, we find that μ_{design} = 3.19/sec, which implies that it requires approximately 300ms to repair an upset. One may intuitively conclude that scrubbing continuously is the best solution to ensure the best availability. However, one of the main drawbacks of this technique is its high power consumption, due to the repeated accesses to the large configuration memory of the FPGA [67]. Thus, it is desirable to perform scrubbing with the minimum required frequency.

In Figure 4.12, we show the availability for different scrub intervals (τ) for a mission time of T = 300 seconds. In PRISM, this property can be formalized as `R{"up_time"} = ?  [C <= T]/T, T = 1 to 300`. Regarding the scrub parameter μ, there are two main things to consider in scrubbing: time when the scrub is triggered and time to repair the bitstreams. For the following experiments, whenever we mention scrub interval, it includes both the time to trigger the scrub and time to repair the bits. We observe that, for τ = 0.5 seconds, the availability is decreasing but stays in the range of five 9s (99.999%) for the whole mission time. On the other hand, for τ = 1 second, the availability drops below five 9s before reaching T = 50 and continues in the range of four 9s. For τ = 1.5 second and 2 seconds, the availability is in the range of four 9s and decreases with time.

In Table 4.2, we show the availability of the system for a mission time of one month for different scrub intervals, τ. We observe that the availability is five 9s for τ = 0.5 second and four 9s for τ = 5 seconds, and for τ = 1000 seconds the availability is only two 9s. The probability that the system will always be operational (with zero failure events) within the first 2 hours of operation can be formalized in PRISM

Table 4.2: Scrub intervals vs reliability and availability

Scrub Interval (s)	**Availability (1 month)**	**Reliability with scrub, T = 2 hr**	**Reliability with scrub & TMR, T = 2 hr**
0.5	0.99999	0.8658	0.9999
5.0	0.99995	0.8658	0.9999
10.0	0.99989	0.8658	0.9999
100.0	0.99899	0.8658	0.9991
1000.0	0.99001	0.8658	0.9918

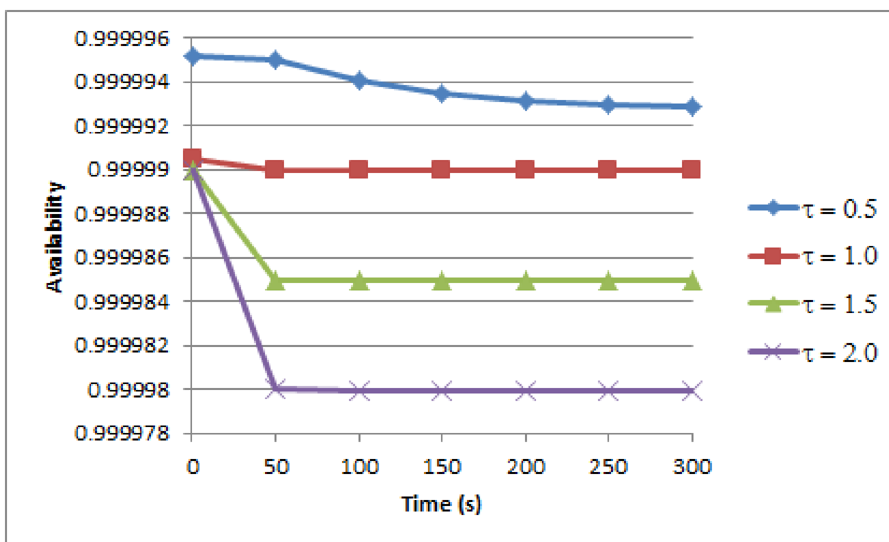

Figure 4.12: Availability for T = 300s

as : `P = ? [G[0, T] ("operational")], T = 7200`, and the results are shown in Table 4.2. Reliability results show that periodic scrubbing has no effect on the reliability. This might seem counter intuitive as one might think that the scrub will drastically increase the reliability. In fact, this result reflects the *memoryless* property of exponential distributions. When an SEU occurs, it might corrupt the system, but if the impact has not arrived yet, the observed time can be reset at any time, that means the coming time of the impact will not be delayed by periodic scrubbing. That is the main reason why the periodic scrubbing will not increase reliability. Availability is defined as the ratio of uptime and total runtime (mission time). That is why, for

repairable systems, system engineers are more interested in availability analysis. The results also show that even though the reliability is not increased, the availability is significantly improved by the scrubbing. To increase the reliability, the designers need to adopt redundancy-based solutions, such as device-level TMR at a cost of at least 3 times area and power overhead, and the results are shown in column 4. The results show how significant reliability improvements are obtained by adopting the TMR-based solution with scrub. We must mention that in such cases, periodic scrub will have an effect on reliability. The reason is, to reach a failure state, it needs at least 2 FU failures (assuming the voter is error free). So after one FU fails, if the scrub interval is short, the system gets back to "all good" state before the second failure occurs. Otherwise, if the wait time is longer for the second scrub, then it might reach the failure state before the scrub is triggered. For example, for a scrub interval of 1000 seconds, we observe that the reliability drops to 0.99.

Such analysis at an early design stage can help a designer to adopt a proper mitigation strategy considering reliability, availability requirement, power and area constraints. For high availability applications, scrubbing alone might fulfill the requirements. Using our methodology, the designer can choose a proper scrub interval to minimize the power requirements for such applications. On the other hand, for reliability oriented applications, a redundancy-based solution is a must. However, it comes with an extra power and area overhead. The area and power overhead can be reduced by applying the other types of TMRs, such as selective TMR [80], however device-level TMR ensures the best reliability [35].

Table 4.3: Verification of availability requirements

Design	Scrub interval (s)	Availability requirement met ?
FIR	0.5	**True**
	1.0	False
	1.5	False
	2.0	False
	2.5	False
	3.0	False
EWF	0.5	True
	1.0	**True**
	1.5	False
	2.0	False
	2.5	False
	3.0	False

4.3.2 Verification

Depending on the mission goal, a spacecraft can have different levels of reliability and availability requirements. For example, a GPS or communication satellite will need a better availability compared to an earth observation satellite. A Mars rover's landing module will require better reliability than the module responsible for taking photos periodically to send back to earth. Finite Impulse Response (FIR) and Elliptic Wave Filter (EWF) filters are widely used in image processing applications and also for sensor's noise reduction. Autonomous landing systems [63] widely use image processing algorithms to find a suitable landing place, hence it is obvious that such a system will require high reliability as the choice of wrong landing place may cause mission failure. We consider a 512-FIR filter for two different applications: in GPS satellites and in a lunar landing module. For the first part of the analysis, we will also analyze a parallel EWF filter structure that has 32 blocks running in parallel, each of which has a separate 16-point Elliptic wave filter [66]. Communication satellites require high availability, usually in the range of five 9s. To be independent of mission time, we calculate the steady state availability to ensure that the availability will maintain

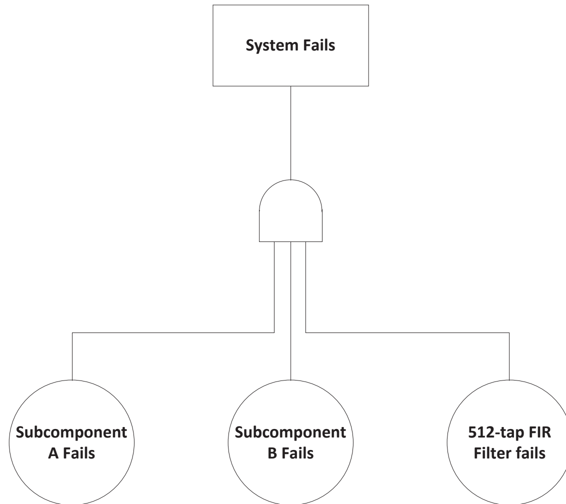

Figure 4.13: Fault tree of the system

that level of service in a long run. So we need to find, *"for a given scrub rate, does the system meet the requirement for five 9s ?"*. Such property can be formalized in PRISM as $S >= 0.99999[\text{"}operational\text{"}]$. To find the appropriate scrub rate, we vary the scrub interval from 0.5 to 3 seconds and verify if the requirement is met. The results are shown in Table 4.3. As we can see from the results, the only scrub interval that meets the requirement for the FIR filter is $\tau = 0.5$ second or less. In contrast, for EWF filter application, the scrub interval can be further delayed up to 1 second.

To verify the DAL requirement, we consider a hypothetical system that uses the FIR filter as a subcomponent in the image processing component of a lunar landing module. As such image analysis needs to be done in real time and a fault during the landing operation will cause a catastrophic failure, so the overall failure probability of such a system must be less than 10^{-9} (DAL-A design) according to Item Design Assurance Level (IDAL) standard. From the fault tree [10] in Figure 4.13, we observe that if any of the subcomponents, namely A, B and the FIR filter fails, the module fails. If subcomponent A and subcomponent B both have a failure probability of 0.0001, then to avoid a catastrophic failure the failure probability of the FIR filter

Table 4.4: Verification of reliability requirements

Scrub interval (seconds)	**DAL-A met (scrub only) (B = 0.0001)**	**DAL-A met (scrub only) (B = 0.001)**	**DAL-A met (scrub & TMR) (B = 0.001)**
0.5	True	False	True
1.0	True	False	True
1.5	True	False	True
2.0	True	False	True
2.5	True	False	True
3.0	True	False	True

must be less than 0.1. It takes around 4 days to reach lunar orbit and touchdown to the moon requires around 11 minutes starting from the descending time. So to be safe, we consider 20 minutes, hence we find out *"For a given scrub interval, what is probability that the FIR filter will fail in last 20 minutes of the flight ? "*. Such property can be formalized in PRISM as `P < 0.1 [F [344400,345600] ("failure")]`. We also evaluate another case, where the failure probability of subcomponent B is 0.001. For this experiment, a system with only scrub and a system with both TMR and scrub, are evaluated while varying the scrub rates (as scrub rate affects the system using TMR with scrub). From the result shown in Table 4.4, we observe that, all the scrub intervals from 1 to 3 seconds meet the DAL-A requirement, whereas in the latter case, if B = 0.001, then it fails to satisfy the DAL-A requirement. However, if TMR is adopted with scrub, even with B = 0.001, the system meets the DAL-A requirement for all the scrub intervals. Such results can help a designer obtain the maximum scrub interval (in this case 3 seconds) to save power.

4.4 Summary

In this chapter we presented a part of our proposed methodology that focuses on the optimization of scrub frequency and verification of DAL compliance. We presented

the reliability and availability models for systems with periodic scrub and TMR developed using conventional Markov chains (instead of semi-Markov). Our analysis using PRISM model checker showed how an appropriate scrub interval (slowest scrub rate) can be found to save power while meeting the dependability requirements. In addition, it was also showed that probabilistic model checking based techniques can be used to verify the high-availability, and the DAL compliance requirements at a high-level. In the next chapter we will explain the last part of our methodology that enables the early analysis of TMR partitioning strategy for optimal partitioning.

Chapter 5

Optimal Partitioning of TMR

Reliability analysis of TMR and related improvements have been studied for a long time and widely reported in many literatures. Compared to that, partitioning of TMR for reliability improvement got less attention from the research community. In this chapter, we illustrate the third part of our methodology that can be utilized to assess the TMR partitioning scheme at early design stages. This part of our work aims to analyze the relationship between the number of TMR partitions, scrub interval and mission time. The proposed formal models of TMR partitioning can handle both equal sized or non-equal sized partitions. In addition, our proposed model can analyze designs capturing both the phenomena, SBUs and MBUs (we assess up to Double Bit Upsets (DBUs)). Note that, multiple bit upsets in different components from a single strike can be classified as Multi-Cell Upsets (MCUs). MCU refers to the flipping of two or more number of bits in the memory array due to one or more radiation particles. However, to be more generic in the rest of this chapter, we will address MCUs as MBUs. The proposed modeling technique is useful to find: (1) the trade-off between the number of partitions and required scrub rate (and vice versa) to meet the target reliability and availability, (2) the optimal number of partitions for designs

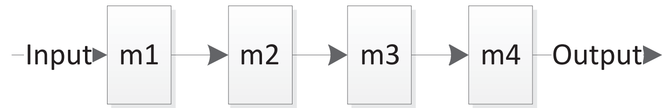

Figure 5.1: Sample unmitigated circuit

prone to both SBUs and DBUs.

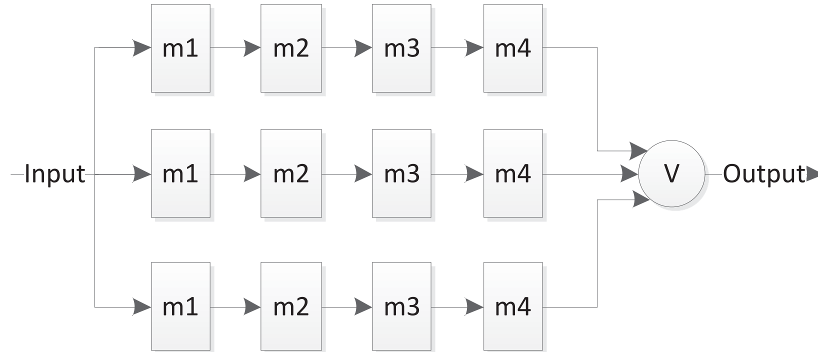

Figure 5.2: TMRed version of the sample circuit

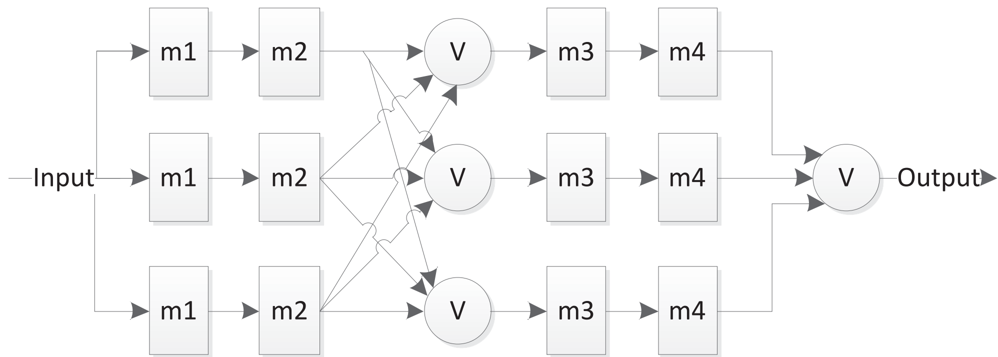

Figure 5.3: The sample circuit with TMR divided into two partitions

5.1 TMR Partitioning

Traditional TMRed design can deal with a single fault at a time. Thus, faults in multiple redundant modules will break the TMR. For illustration, Figure 5.1 shows

a sample circuit (each box represents a module) and Figure 5.2 shows the TMR implementation of the sample circuit in the traditional way. While designing a system with traditional TMR, the components are triplicated and a majority voter is placed at the output of the circuit. The voter can provide correct outputs even if one of the branches (or domains) of the TMR is faulty. Figure 5.3 shows the same system implemented with partitioned TMR (as suggested in [75]). In terms of dependability, each partition can be considered as a separate entity. This circuit will only fail if two or more domains in the same partition are affected by one or more faults. For example, upsets in module $m2$ in domain two (second row) and $m3$ in domain one (first row) will break a traditional TMR system, whereas it will get successfully masked in a system with partitioned TMR.

5.2 Methodology for TMR partitioning analysis

Figure 5.4 presents the proposed methodology. Once the CDFG is extracted, depending on the resource, performance or area constraint for hardware implementation, it can be scheduled using the appropriate scheduling algorithm. Since, scheduling is out of the scope of this work, we assume a fully parallel implementation of the CDFG for high performance. However, it is worth mentioning that, the methodology will work irrespective of the scheduling approach. Depending on the number and size of partitions defined by the user, each domain in each partition can be represented as one or a collection of nodes (nodes in the graph represent a basic operation such as add, multiply, etc.). Each node can be implemented as a component in the FPGA. We use the component characterization library to estimate the failure rate of each component as mentioned in the earlier chapters. With this approach, the failure rate calculation of each domain becomes straightforward. Indeed, the failure rate of a TMR domain in

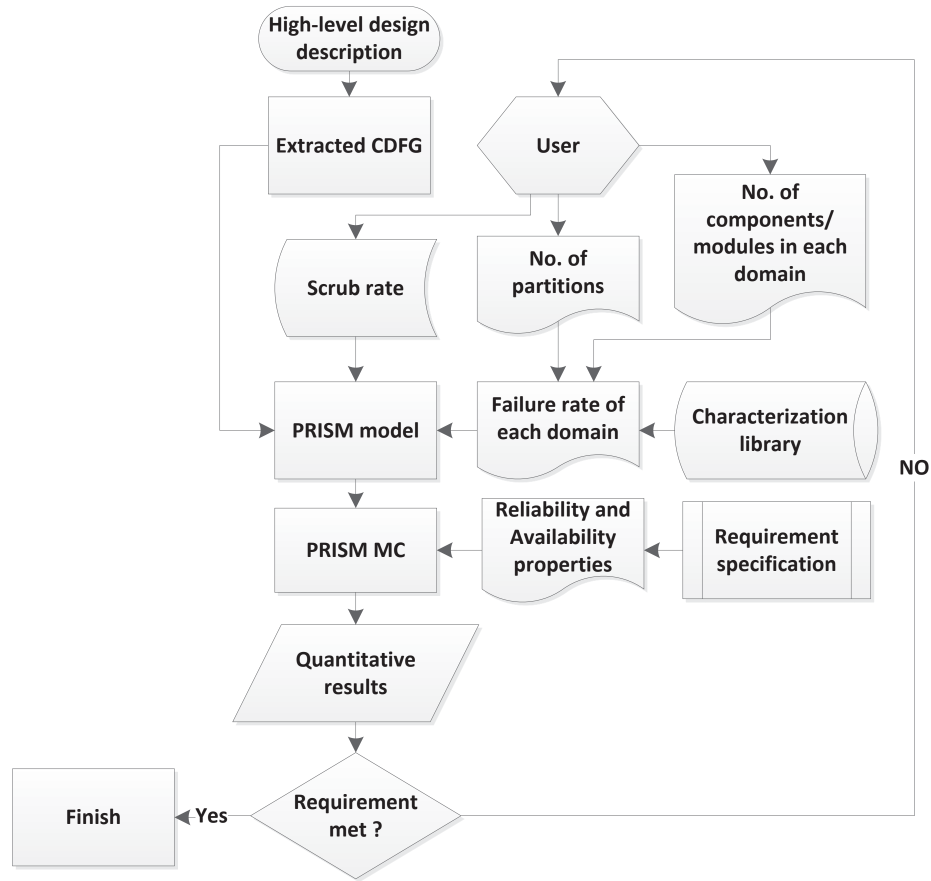

Figure 5.4: Methodology for optimal TMR partitioning

a specific partition is equal to the total failure rate of the components in that domain. Based on the calculated failure rate of each domain, the number of partitions and the user defined scrub rate, a PRISM model is then built. The PRISM model checker then automatically converts the PRISM model to an equivalent CTMC representation. Different reliability and availability properties are then verified to check if the design meets the requirements. The PRISM model checker provides quantitative results. If the requirements are not met, the number of partitions or the scrub frequency is then modified, and the analysis is performed again.

For illustration, in Figure 5.5 the partitioning of a CDFG representing an 8-tap FIR filter is shown. All the domains in partition-1 have four multipliers and three

adders. On the other hand, each of the domains that are part of partition-2 contains four multipliers and four adders. Note that we use the same characterization library that was introduced in chapter 3 to calculate the failure rate of the TMR domains. The proposed methodology is generic enough to be used with a different characterization library with more precise and accurate data, without any major changes. The failure rate for a component can be calculated using the following equation :

$$\lambda_{component} = \lambda_{bit} \times Number\ of\ critical\ bits$$

For our experiments, $\lambda_{bit} = 7.31 \times 10^{-12}$ SEUs/bit/sec for Higher Earth Orbit. So, the failure rate of a single domain in partition-1 is the summation of total failure rate from 4 multipliers and 3 adders. The failure rate of partition-1 is then just the failure rate of a domain in partition-1 multiplied by 3. It is important to mention that an SEE can cause either an SBU or MBU. Hence, λ_{domain} need to be adjusted accordingly. The simplest way is to multiply the λ_{domain} with the SBU and MBU coefficient, α_{SBU} and α_{MBU} respectively.

5.2.1 Markov Modeling of Single Bit Upsets (SBUs)

Before modeling the TMR partitions, we describe the traditional CTMC model of a TMR system with scrubbing (no partition) that is shown in Figure 5.6. Each node in the model denotes the current state of the system: state 3 represents the state in which all domains are operating correctly (all the modules are fault-free), state 2 represents a state where one out of three domains is operating incorrectly (in one of the domains at least one of the modules is faulty), but the output is still not erroneous, and state 1 represent a *failure state* in which two or more domains are operating incorrectly (more than one module is faulty in two or more domains). The failure rate λ represents the failure rate of each domain in the TMR and μ represents the scrub rate. Since all the

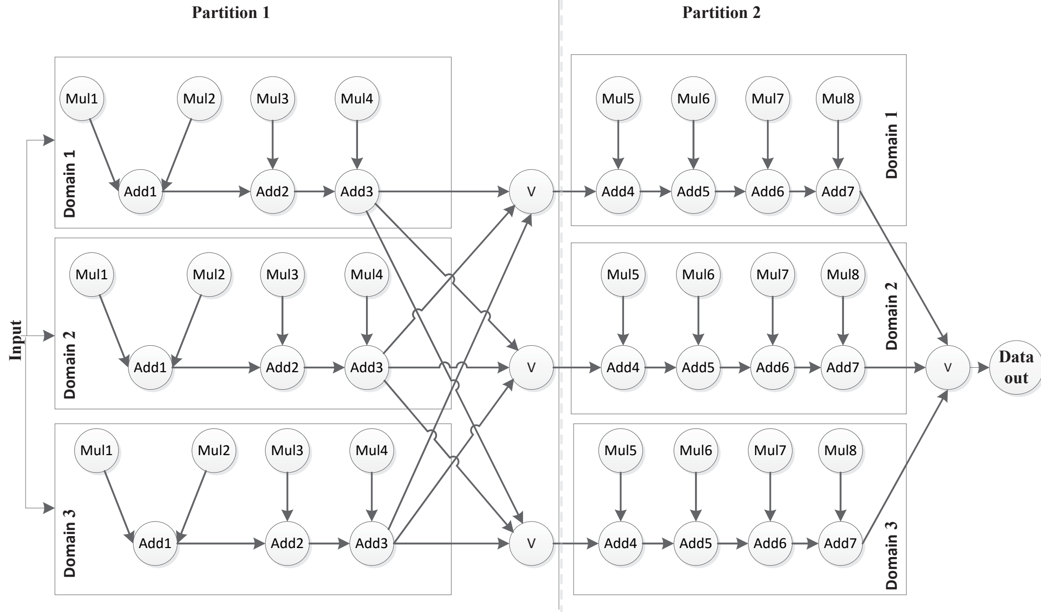

Figure 5.5: TMR implementation of the CDFG of an 8-tap FIR filter with two partitions

domains in TMR have an equal failure rate, the transition rate from state 3 to state 2 can be depicted by 3λ (the sum of failure rates of each of the individual domains).

As a first step in our modeling, we modify the TMR Markov model presented in [75] by adding some extra (self) transitions to represent some possible scenarios. Since we consider periodic blind scrub, so from state 3 the system can either go to state 2, or get scrubbed with a defined scrub rate. The scrub transition in state 3 is represented by a self-transition with a rate μ. From state 2, the system has three options: (1) the system can get scrubbed and go back to state 3; (2) another module in a fault-free domain of the TMR can fail — which will lead the system to state 1 with a transition rate 2λ or (3) SEUs (causing an SBU) can hit the same module or another module in the same domain that failed earlier, in which case, the system will stay in the same state with a transition rate λ. Once the system enters the state 3, which is a failed state, it will remain in that state until the system gets scrubbed eventually and comes back to state 3 with the scrub rate μ. The assumptions for the

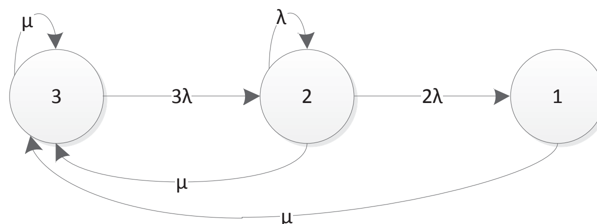

Figure 5.6: Markov model of TMR with repair

model can be defined as follows:

Assumption 1: Each module(components) in the TMR may fail independently. Since the modules in a domain are non-redundant, so a module failure represents the failure of the respective domain. The time-to-failure due to a configuration bit flip (either SBU or DBU) is exponentially distributed. The exponential distribution is commonly used to model the reliability of systems where the failure rate is constant. The *scrub* interval is assumed to follow an exponential distribution as well, with a rate, $\mu = 1/\tau$, where τ represents the scrub interval.

Assumption 2: The design employs the blind scrubbing technique.

Assumption 3: Only one module can fail at a time due to an SBU.

Assumption 4: The majority voters in TMR are assumed to be error free.

Assumption 5: All the states in the CTMC model can be classified into two types: *operational*, where at most one or no domain in any of the partitions are faulty; and *failed*, which means there is at least one partition, where more than one domain is faulty. In PRISM, a *formula* can be used to classify such states.

Before we go to the second step of modeling, we formalize the Markov model of the TMR in Figure 5.6. As mentioned earlier, we consider a CTMC as a finite transition system (S, R, L). The set of states can be denoted by $S = \{3,\ 2,\ 1\}$. In state $i \in S$, i represents the number of domains that are healthy. The rate matrix R is depicted on the edges, for example, $R(3,2) = 3\lambda$, $R(2,1) = 2\lambda$, $R(2,2) = \lambda$, $R(2,3) = \mu$, $R(3,3) = \mu$ and $R(1,3) = \mu$. Let the atomic propositions *up* and *down* denote that the system works correctly or not respectively. Since it is a TMR system, 2 out of 3 domains need to be working at a time. Then, $L(3) = L(2) = up$, i.e., the states 3 and 2 satisfy the atomic proposition *up*. Similarly, $L(1) = down$, i.e., the states 1 satisfy the atomic proposition *down*.

In our modeling, each TMR partition is modeled as a separate CTMC. This means that each of the partitions has a separate CTMC that is equivalent to the model shown in Figure 5.6. Hence, a system with N partitions can be defined by a set:

$$P = \{P_0, P_1, ..., P_N\}$$

Here each $P_i \in P$ represents a CTMC. If the system is divided into N partitions, then its final model is defined by the parallel composition ($\|$) of all the CTMCs from the partitions:

$$M = \{P_0 \parallel P_1 \parallel .... \parallel P_N\}$$

The total state space of the model M will be the cross product of states in all the partitions. The PRISM code for two partitions is shown in Figure 5.7. Each module in the PRISM code represents a partition in the TMR. `num_P1_M` and `num_P2_M` define the number of domains in each partition. In TMR the number of domains is 3, so both of these parameters need to be initialized with a value of 3. `Lambda_P1`

```
1 module Partition-1
2 P1 : [1..num_P1_M] init num_P1_M;
3 // num_P1_M = 3 means all modules working fine, num_P1_M =
    2 means one of the module is faulty, num_P1_M means TMR
    failed
4 [] (P1  > 1) -> P1*lambda_P1 : (P1'=P1-1);
5 // failure of a modules due to SBU
6 [] (P1 = 2) -> lambda_P1 : (P1'= 2);
7 // the same module that already failed can fail again
8 [dbu] (P1  > 1) -> (P1*lambda_P1D+P2*lambda_P2D) : (P1'=P1
    -1);
9 // failure of a modules in either partitions due to a DBU
10 [rep] (P1 <=  num_P1_M) -> repair : (P1'=num_P1_M);
11 // scrubbing can fix all the modules
12 endmodule
13
14 module Partition-2
15
16 P2 : [1..num_P2_M] init num_P2_M;
17 // num_P2_M = 3 means all modules working fine, num_P2_M =
    2 means one of the module is faulty, num_P2_M = 1 means
    TMR failed
18 [] (P2  > 1) -> P2*lambda_P2 : (P2'=P2-1);
19 // failure of a modules
20 [] (P2 = 2) -> lambda_P2 : (P1'= 2);
21 // the same module that already failed can fail again
22 [dbu] (P2  > 1) -> 1 : (P2'=P2-1);
23 // synchronization with partition-1 to represent a DBU
24 [rep] (P2 <=  num_P2_M) -> 1 : (P2'=num_P2_M);
25 // scrubbing can fix all the modules
26 endmodule
27
28
29 formula fail  =   (P1 = 1) | (P2 = 1) ;
30 formula operational = !fail;
```

Figure 5.7: PRISM Code of a TMR with two partitions (SBU and DBU)

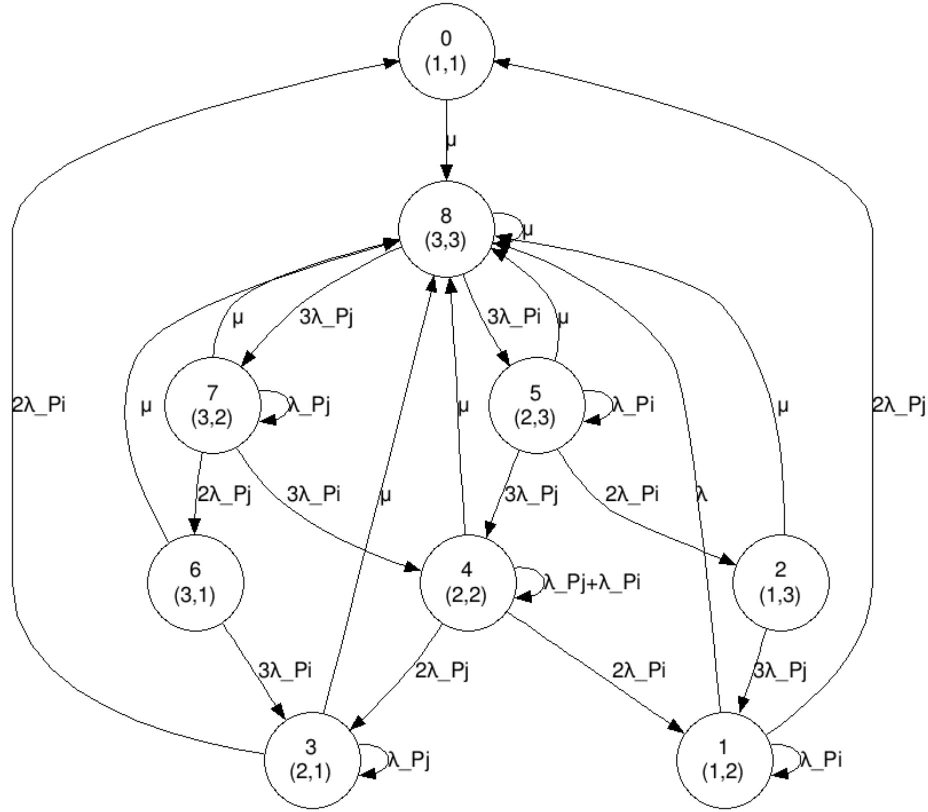

Figure 5.8: SBU Markov model of TMR with two partitions

and `Lambda_P2` defines the failure rate of a domain in partition-1 and partition-2 respectively. The `repair` parameter in the model denotes the user defined scrub rate. When the FPGA is scrubbed, SEUs in all the modules of each partition are repaired, the synchronization label `[rep]` is used to model this phenomenon. Formula `fail` and formula `operational` define that the system will fail if any of the partitions have more than one faulty domains; otherwise the system is operational. Line 8 and Line 22 are related to MBU modeling, hence can be ignored for this part of the modeling.

Once the final model is built by the parallel composition of PRISM modules using the PRISM model checker tool, quantitative analysis can be performed automatically to analyze different design options. Note that, PRISM has a feature known

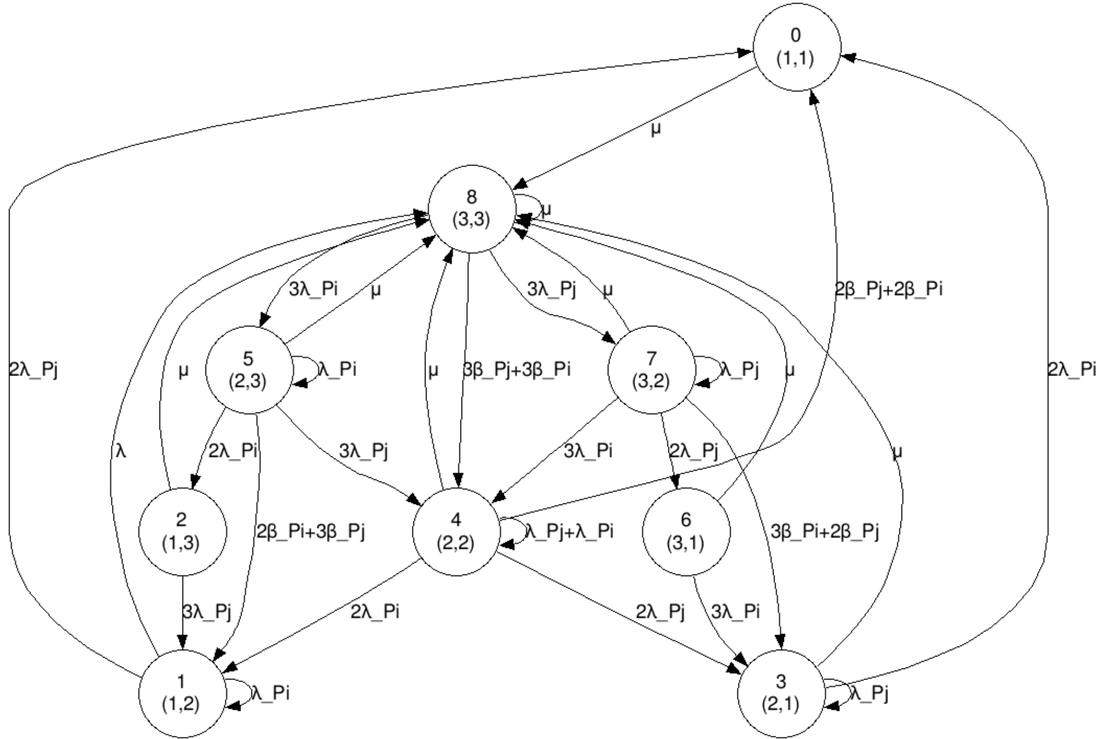

Figure 5.9: DBU Markov model of TMR with two partitions

as *Module renaming*, which can be used to extend this model to N number of partitions with minimal effort. The equivalent CTMC representation of this PRISM code is shown in Figure 5.8 where λ_{Pi} and λ_{Pj} represent `Lambda_P1` and `Lambda_P2` in the code.

5.2.2 Markov Modeling of Multiple Bit Upsets (MBUs)

An SEU can flip two or more number of bits simultaneously in the FPGA configuration bitstream. Since upsets of more than two simultaneous bits are not very common (still occurs though with a very low probability), hence in this thesis, we limit our modeling to Dobule-bit Upsets (DBUs). An SEU that causes a DBU may invoke failures in multiple TMR domains simultaneously. This situation is more common in a harsh radioactive environment such as in outer space. Modeling of a combined model

that include both SBU and DBU in a single Markov model requires an additional assumption:

Assumption 1: DBUs can occur at a specified rate and could effect two TMR domains simultaneously in two separate partitions. In other words, this assumption means that our modeling restricts failure of two simultaneous domains in the same partition.

Figure 5.9 shows the Markov model that considers the effect of both: SBUs and DBUs. In the model, β_{Pi} and β_{Pj} represent the DBU rate of a domain in the first and second partition respectively. For example, in state 8 both the partitions are working fine. However, if one of the domains in the partitions encounters an SBU, then the system can move to either state 5 or state 7, depending on the location of the domain. Also, if the system is in state 8, and a DBU occurs in any domain of either partition, it will trigger another domain failure in the other partition simultaneously. This leads a path from state 8 to state 4 with the rate $3 * \beta_{Pj} + 3 * \beta_{Pi}$.

Modeling DBU in PRISM is quite a challenge since it needs the use of synchronization of associated commands in different modules to represent a simultaneous failure due to a DBU. Line 8 and Line 22 in Figure 5.7 shows the PRISM codes (`Lambda_P1D` and `Lambda_P2D` depict the DBU rate of domains in the corresponding partition) that need to be added to model both SBU and DBU for two partitions. For increased number of partitions, a number of extra synchronization commands are added to each module. For instance, in the case of 4 partitions, each module in the PRISM code will have three extra commands for synchronization of DBU failures.

5.3 Quantitative Analysis of an FIR Filter using PRISM

To illustrate the applicability of our approach, we analyze an 8-bit 64-tap FIR filter (the target platform is Xilinx Virtex-5 SRAM-based FPGA) using both, the SBU model and the combined model (that considers both SBUs and DBUs) for a different number of partitions. An N-tap discrete finite impulse response (FIR) filtering can be expressed as following:

$$y[n] = \sum_{i=0}^{N-1} x[n-i] \cdot h[i]$$

$x[\ ]$, $y[\ ]$ and $h[\ ]$ are the input samples, output samples and the filter coefficients respectively. All the experiments are conducted for a mission time of 1 month. Since SEUs can cause both, either SBUs or DBUs, for the combined model, it was assumed that 90% ($\alpha_{SBU} = 0.9$) of the SEUs will cause SBUs and 10% ($\alpha_{MBU} = 0.1$) of them will cause DBUs. On the other hand, for the SBU model, it was assumed that all the SEUs will only cause SBUs. Since the model is parametric, any other values for scaling the SBU and DBU rates can be used. Table 5.1 shows the model generation statistics for both models. For the analysis, four design options are analyzed using our methodology, starting from no partition up to eight partitions. According to the assumption 1 in MBU modeling, for DBU analysis we need at least two partitions. So *no partition* option is ignored for the combined model. We use the PRISM model checker version 4.1 to analyze the reliability and availability properties for each of them.

Figure 5.10 shows the relationship between the reliability and number of partitions in the design for different scrub intervals using the SBU model. Reliability of a

Table 5.1: Model construction statistics

No. of partitions	**No. of states**	**No. of Transitions SBU only model**	**No. of Transitions Combined model**
0	3	6	N/A
2	9	26	30
4	81	362	578
8	6561	478858	129506

system (or component) is defined as the probability that the system performs correctly for a given period of time, from zero (t_0) to t_1, given that the system or the component was functioning correctly at t_0. In PRISM, this property can be formalized in CSL as `P = ? [G[0,T] operational], T = 1 month`, and we evaluate this property for different scrub intervals starting from 15 minutes up to 4 hours. For all the design options with different scrub intervals, the reliability decrease when the scrub interval increases. However, designs with more partitions show significant improvements in reliability even with the same scrub interval. For example, if the scrubbing interval is 15 minutes, the design with no partition has a reliability of 0.71 only. In contrast, the design with two, four and eight partitions has a reliability of 0.81, 0.90 and 0.94 respectively. TMR increases the area and power consumption by a factor of 300% as a result of replications. More frequent scrub in such cases will consume more power that might not be appropriate for most space applications. For such circumstances, increasing the number of partitions can offer a good solution instead of a more frequent scrub strategy. For example, if the designer is targeting the reliability more than 0.80, and if the design has no partitions (or less number of partitions), then the designer may think to adopt a more frequent scrubbing strategy (less than an hour, in order of seconds or milliseconds) to meet the requirement. Instead of adding such power burden on the system, the designer may adopt TMR with 2, 4 or 8 partitions, which will require scrubbing once per 15 minutes (comparatively less power consumption)

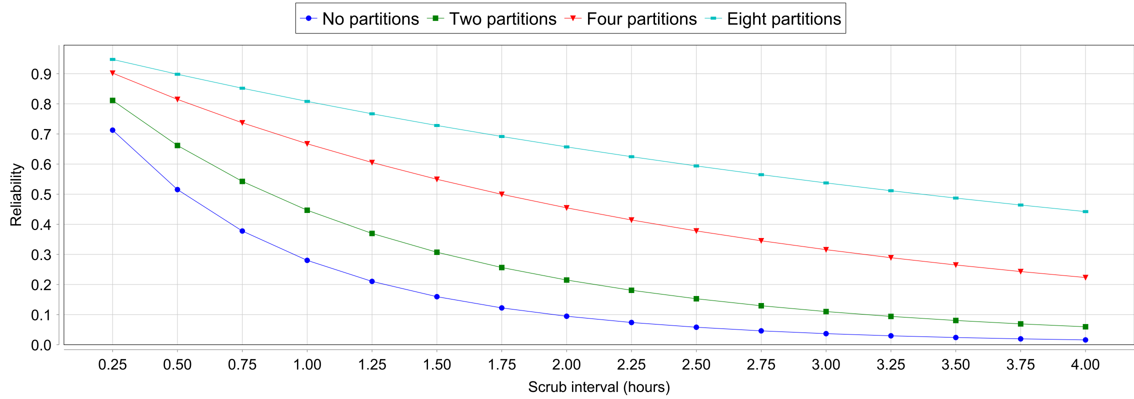

Figure 5.10: SBU Reliability

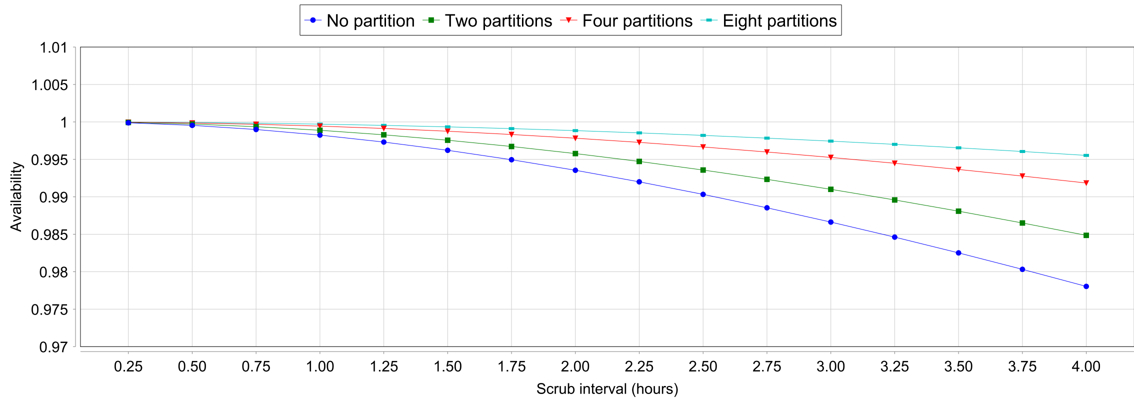

Figure 5.11: SBU Availability

and will also meet the requirement. Note that the design option with eight partitions provides a reliability of 0.8 even for a delayed scrub of 1 hour. Using this approach a designer can qualitatively assess the number of partitions required to meet the design requirements for a given scrub rate or vice versa.

Availability is defined as the ratio of time the system or component operates correctly (system uptime) to its entire mission time. Using the SBU model, Figure 5.11 shows the availability of the design for different scrub intervals and a different number of partitions. In PRISM, this property can be formalized in CSL as `R{"up_time"} = ? [ C<=T ]/T, T = 1 month`. The design with only no partition offers the availability of 5 nines (0.99999) for the scrub interval of 15 minutes which drops up to 1 nine (0.97)

with increased scrub interval of 4 hours. Compared to this, all the other options with TMR partitioning offers improved availability. For instance, for the scrub interval of three hours, the design with no partition offers only 98% availability, whereas the rest of the design options with partitioning offers the availability of more than 99%. Most of the communication satellites targets more than 99% availability. In such cases, if the power constraint restricts the designer not to increase the scrub interval, then the increasing the number of partitions may offer a solution.

A major observation from these analyses is, when the scrub interval is smaller (frequent scrub), the number of partitions plays a major role increasing the reliability of a system. However, even for a delayed scrub, the improvement is noticeable enough. In other words, the graphs show a trend that, the more the number of partitions (which means smaller domains), the less frequent scrub will be required to meet a target reliability. Less number of partitions (larger domain size) will require more frequent scrub to meet a target reliability requirement. For availability, the number of partitions plays a significant role for longer scrub intervals. For frequent scrub intervals, the number of partitions increases the availability to a minimal level, but for longer scrub intervals the improvement of availability with the number of partitions is quite significant. Such early analysis on the high-level design description will allow a designer to perform the analysis before the actual implementation of the system considering the design constraints such as power. Using such methodology a designer can find the trade-off between the number of partitions and the required scrub interval that will meet the design requirements, and also reduce the design effort, time and cost.

For the second part of our analysis, we evaluated the same reliability and availability properties using the combined model. The results are shown in Figure 5.12

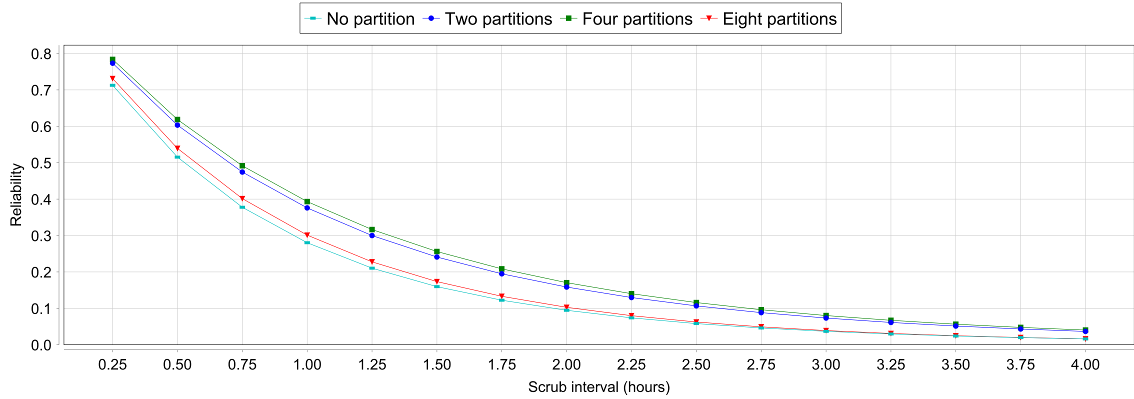

Figure 5.12: Combined Reliability

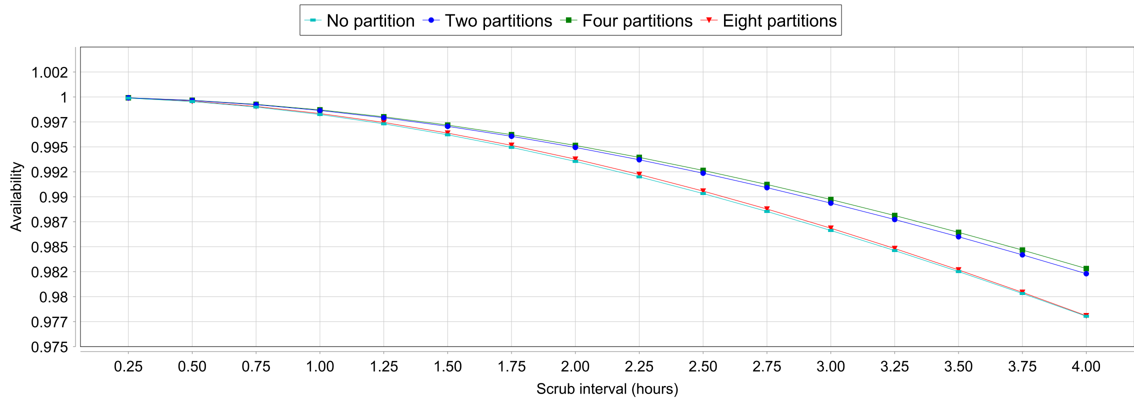

Figure 5.13: Combined Availability

and Figure 5.13. We observe that for both the properties, all the design options with two, four and eight partitions increases the reliability and availability compared to the design with no partition. The reliability and availability are improved to a minimal level when the number of partition increases from two to four. However, it clearly shows that the TMR with eight partitions is less reliable and available compared with the TMR with two and four partitions, which contrast the SBU only model. From this, we can conclude that the optimal number of partitions for this model is four if the design is prone to both SBUs and DBUs. This is due to the fact that, after partitioning the design into a certain number of partitions, any extra partition added to the design also increases the probability of DBUs causing a system-wide failure.

Similar findings for Domain Crossing Event (DCE) to suggest an optimal number of partitions was reported in [46] using fault injection. In our case, we are able to perform such analysis at the early design stage.

5.4 Summary

In this chapter, we presented the last part of our proposed methodology, that is the formal modeling and analysis of SBUs and MBUs using probabilistic model checking. Our analysis shows that increasing the number of TMR partitions can reduce the frequency of scrubbing, which will eventually result in less power consumption. However, if the design is prone to both SBUs and DBUs, then there exists an optimal number for partitioning. Using the proposed methodology, designers can assess the number of partitions, or the scrub frequency required to meet the design requirement at early design stages. To validate these claims, we have shown the results of our analysis for a 64-tap FIR filter case study. The results showed how the increased number of partitions can cope with the less frequent scrubs and vice-versa.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

This thesis proposed a methodology for high-level dependability and performability analysis of SEU prone SRAM-based FPGA designs. We illustrated how the probabilistic model checking technique can be used to analyze designs at early stages for space applications. The proposed methodology allows designers to perform three different types of analysis: design options analysis for performability optimization, scrub interval optimization with design assurance level analysis, and optimal partitioning of TMR. From the high-level design description, the CDFG is extracted first. Depending on the required analysis, a branch of the methodology is chosen. To follow this process, the CDFG is then modeled in the PRISM model checker with the help of other necessary information for further analysis. To demonstrate the applicability of the proposed methodology, we presented case studies on benchmark DSP circuits. To our knowledge, this is the first work in this area that shows the use of probabilistic model checking for the high-level dependability analysis of SEU prone FPGA-based systems.

For the first part of the methodology, various design options were analyzed, each of which was modeled using a Markov reward model. Such modeling captures the possible failures, fault detection coverage and repairs (scrubbing, rescheduling, cold-spare) possible in a high-altitude radiation environment. Afterwards, a wide range of properties were exhaustively and automatically verified to evaluate the design options, regarding throughput, area, and dependability. The obtained results showed how coverage can impact the dependability and performability based on mission time, scrub interval and adopted mitigation strategy. Interestingly, we found that coverage alone can not guarantee high reliability. Also, it was observed that in some cases, rescheduling can serve as a better mitigation technique when compared to the redundancy-based solutions.

As the second part of the methodology, we showed how our method can be used to verify the high-availability requirements and the design assurance level compliance at high-level. Since in classical Markov chains the delay is exponentially distributed, we utilized the Erlang distribution to accurately model the scrub intervals. The obtained modeling results showed how an appropriate scrub interval (slowest scrub rate) can be found to save power while meeting the dependability requirements.

Finally, in the last part of our methodology, we analyzed the scrubbed partitioned TMR systems for optimal partitioning. Our modeling is novel compared to others since the proposed model can evaluate both equal and non-equal sized partitions. Also, instead of concentrating only on single-bit upsets, the modeling of double-bit upsets was also introduced. Based on the obtained results, we concluded that for designs that are prone to both single and double bit upsets, an optimal number of TMR partitions can be found. Using our method, we were able to find the optimal partitioning at early design stages instead of adopting the fault injection or

beam testing approaches.

From system biology to gate-level circuit analysis, from robotics to chemical reaction — probabilistic model checking has already been used in many different domains. During this research work, we realized that probabilistic model checking has great applicability in the area of SEE analysis. The randomness of the radioactive environment, with different fault mitigation techniques and their associated trade-offs, makes it an ideal problem statement that be analyzed automatically using state-of-the-art probabilistic model checker tools. We analyzed designs at a high-level, hence the state explosion problem was not encountered during our research. The PRISM model checker can handle up to 10^{10} states. Due to this fact, it is possible to handle even larger designs if the CDFG to PRISM modeling part is fully automated.

6.2 Future Work

Dependability analysis is one of the most important phases in the design flow for complex systems. For the case of space applications, an early analysis may help to verify the design requirements in early design phases. In addition to increasing the designer's confidence in the design, such analysis may also reduce the associated cost, time and design effort. This thesis lays the ground for a promising approach for the early dependability analysis of SEU prone FPGA-based applications. Building on the proposed methodology and verification results presented in this thesis, several extensions can be explored to further strengthen the proposed method. Some future research directions are outlined as follows.

- TMR techniques are traditionally used to mitigate SEUs, but with an overwhelming amount of extra area and power. In [42], the authors proposed a

framework for reconfigurable fault tolerance that enables designers to dynamically change the amount of redundancy for fault mitigation. Such a technique can be modeled using the concept of the *Phased-mission Markov model* to estimate the achieved performability gains. The inclusion of such dynamic mitigation models in our proposed methodology can be an interesting extension. This will also help to find how effective the rescheduling-based fault mitigation techniques are if adopted as an option in the dynamic fault mitigation strategy.

- In this thesis, we explored the effect of SEUs at a high-level. However, it is possible to include other fault models using our method, such as analyzing design failures due to aging, electromigration, hot electron effects, and Negative-Bias Temperature Instability (NBTI) and Single-Event Functional Interrupts (SEFI).

- For the design options analysis, we considered the area and throughput metric while modeling performability. In an FPGA design, even if a component fails, it will still consume power (providing the wrong results until the scrub fixes the SEUs). Using our method, it is possible to model such a phenomenon by adding the component's power consumption as a reward in the Markov Reward model to further optimize the power consumption.

- In this thesis, we used Erlang distribution to model the blind scrub. An interesting extension of this work can be the use of Erlang distribution for modeling the partial reconfiguration (read-back scrubbing). In that case, the modeling of two discrete time intervals are required: time to detect the fault and time to fix the fault. In addition, the TMR voter's fault coverage can also be added to the model for better accuracy. It is worth mentioning that we have already applied a similar approach for accurate reliability, availability and maintainability

analysis of a single satellite system [Bio-Cf3].

- In the part of the methodology where the TMR partitions were analyzed, only the single and double bit upsets were considered. It is worth mentioning that due to the rapid decrease of the transistor size, three or more bit upsets are also not uncommon these days. Using a similar approach to the one we introduced in optimal TMR partitioning, it is possible to extend our model to handle upsets in a larger number of bits. Another interesting future work could be to include the partial reconfiguration (read-back scrubbing) in the model in order to explore the effect of unreliable voters in the TMR partitions.

Bibliography

[1] Virtex-5 FPGA configuration user guide, Xilinx UG191, 2012.

[2] Configuration Devices for SRAM-Based LUT Devices, Altera Corporation, 2012.

[3] Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices, Altera Corporation, 2014.

[4] N. Abbasi. *Formal Reliability Analysis using Higher-Order Logic Theorem Proving*. PhD thesis, Concordia University, Montreal, Quebec, Canada, 2012.

[5] P. Adell, G. Allen, G. Swift, and S. McClure. Assessing and mitigating radiation effects in Xilinx SRAM FPGAs. In *Radiation and Its Effects on Components and Systems (RADECS), 2008 European Conference on*, pages 418–424, 2008.

[6] G. Asadi and M.B. Tahoori. An accurate SER estimation method based on propagation probability. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 306–307 Vol. 1, March 2005.

[7] G. Asadi, M.B. Tahoori, B. Mullins, D. Kaeli, and K. Granlund. Soft error susceptibility analysis of SRAM-based FPGAs in high-performance information systems. *Nuclear Science, IEEE Transactions on*, 54(6):2714–2726, 2007.

[8] C. Baier, J-P. Katoen, and H. Hermanns. Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In *Proceedings of the 10th International Conference on Concurrency Theory*, CONCUR '99, pages 146–161, London, UK, UK, 1999. Springer-Verlag.

[9] M. D. Beaudry. Performance-related reliability measures for computing systems. *Computers, IEEE Transactions on*, C-27(6):540–547, 1978.

[10] D. J. Bechta, S. J. Bavuso, and M. A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *Reliability, IEEE Transactions on*, 41(3):363–377, 1992.

[11] Peter A Beerel, Recep O Ozdag, and Marcos Ferretti. *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, NY, USA, 1st edition, 2010.

[12] Express: High-Level Synthesis Benchmarks. `http://express.ece.ucsb.edu/benchmark/`.

[13] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K.A. Label, M. Friendlich, H. Kim, and A. Phan. Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis. *Nuclear Science, IEEE Transactions on*, 55(4):2259–2266, 2008.

[14] P. Bernardi, M.S. Reorda, L. Sterpone, and M. Violante. On the evaluation of SEU sensitiveness in SRAM-based FPGAs. In *On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International*, pages 115–120, July 2004.

[15] C. Bolchini, A. Miele, C. Sandionigi, N. Battezzati, L. Sterpone, and M. Violante. An integrated flow for the design of hardened circuits on SRAM-based

FPGAs. In *Test Symposium (ETS), 2010 15th IEEE European*, pages 214–219. IEEE, 2010.

[16] B. R. Borgerson and R. F. Freitas. A reliability model for gracefully degrading and standby-sparing systems. *IEEE Trans. Comput.*, 24(5):517–525, May 1975.

[17] Teresa M Braun. *Satellite Communications payload and system*. John Wiley & Sons, 2012.

[18] Brendan Bridgford, Carl Carmichael, and Chen Wei Tseng. Single-event upset mitigation selection guide. *Xilinx Application Note, XAPP987 (v1. 0)*, 2008.

[19] P.M. Broadwell. *Response Time as a Performability Metric for Online Services*. Report (University of California, Berkeley. Computer Science Division). Computer Science Division, University of California, 2004.

[20] F. Brosser and E. Milh. SEU Mitigation Techniques for Advanced Reprogrammable FPGA in Space, Master thesis, Chalmers University of Technology, 2014.

[21] C. Carmichael. Triple Module Redundancy Design Techniques for Virtex FPGAs (XAPP197 v1.0.1), Xilinx Corporation, 2006.

[22] D. Chen and K. S. Trivedi. Closed-form analytical results for condition-based maintenance. *Reliability Engineering & System Safety*, 76(1):43–51, Elsevier, 2002.

[23] Dongyan Chen and Kishor S Trivedi. Analysis of periodic preventive maintenance with general system failure distribution. In *Dependable Computing, 2001. Proceedings. 2001 Pacific Rim International Symposium on*, pages 103–107. IEEE, 2001.

[24] Elham Cheshmikhani and Hamid R Zarandi. Probabilistic analysis of dynamic and temporal fault trees using accurate stochastic logic gates. *Microelectronics Reliability*, 2015.

[25] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[26] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[27] Nicolas Coste, Holger Hermanns, Etienne Lantreibecq, and Wendelin Serwe. Towards performance prediction of compositional models in industrial gals designs. In *Computer Aided Verification*, pages 204–218. Springer, 2009.

[28] P. Coussy, C. Chavet, P. Bomel, D. Heller, E. Senn, and E. Martin. GAUT: A high-level synthesis tool for DSP applications. In P. Coussy and A. Morawiec, editors, *High-Level Synthesis*, pages 147–169. Springer Netherlands, 2008.

[29] T. K. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck. Solving semi-Markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560–574, 1999.

[30] Aldous David and Shepp Larry. The least variable phase type distribution is erlang. *Stochastic Models*, 3(3):467–473, 1987.

[31] T.A. DeLong, D.T. Smith, and B.W. Johnson. Dependability metrics to assess safety-critical systems. *Reliability, IEEE Transactions on*, 54(3):498–505, Sept 2005.

[32] G. Aigner et al. The SUIF program representation. http://suif.stanford.edu/suif/suif2/index.html, January 2010.

[33] M. Fazeli, S. N. Ahmadian, S. G. Miremadi, H. Asadi, and M. B Tahoori. Soft Error Rate Estimation of Digital Circuits in the Presence of Multiple Event Transients (METs). In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[34] T.W. Fry and S. Hauck. Hyperspectral image compression on reconfigurable platforms. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, pages 251–260, 2002.

[35] S. Habinc. Functional Triple Modular Redundancy (FTMR). VHDL Design Methodology for Redundancy in Combinatorial and Sequential Logic. *Gaisler Research, Design and Assessment Report (Version 0.2)*, 2002.

[36] O. Hasan. *Formal Probabilistic Analysis using Theorem Proving.* PhD thesis, Concordia University, Montreal, Quebec, Canada, 2008.

[37] B. R. Haverkort, L. Cloth, H. Hermanns, and J. Katoen. Model Checking Performability Properties. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 103–112. IEEE Computer Society, 2002.

[38] P. Hazucha and C. Svensson. Impact of CMOS Technology Scaling on the. Atmospheric Neutron Soft Error Rate. *Nuclear Science, IEEE Transactions on*, 47(6):2586–2594, Dec 2000.

[39] I. Hong, M. Potkonjak, and R Karri. Heterogeneous BISR-approach using system level synthesis flexibility. In *Design Automation Conference 1998. Proceedings of the ASP-DAC '98. Asia and South Pacific*, pages 289–294, 1998.

[40] R. Huslende. A combined evaluation of performance and reliability for degradable systems. In *Proceedings of the 1981 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 157–164. ACM, 1981.

[41] ISOGraph. `http://www.isograph-software.com`.

[42] Adam Jacobs, Alan D. George, and Grzegorz Cieslewski. Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space. In *19th International Conference on Field Programmable Logic and Applications, FPL 2009, August 31 - September 2, 2009, Prague, Czech Republic*, pages 199–204, 2009.

[43] D. N. Jansen, J. P. Katoen, M. Oldenkamp, M. Stoelinga, and I. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *Hardware and Software: Verification and Testing*, pages 69–85. Springer, 2008.

[44] Stephen B Johnson, Thomas Gormley, Seth Kessler, Charles Mott, Ann Patterson-Hine, Karl Reichard, and Philip Scandura Jr. *System health management: with aerospace applications*. John Wiley & Sons, 2011.

[45] R Karri and A. Orailoglu. High-level synthesis of fault-secure microarchitectures. In *Design Automation, 1993. 30th Conference on*, pages 429–433, 1993.

[46] F Lima Kastensmidt, Luca Sterpone, Luigi Carro, and M Sonza Reorda. On the optimal design of triple modular redundancy logic for SRAM-based FPGAs.

In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*, pages 1290–1295. IEEE Computer Society, 2005.

[47] Jan Kastil, Martin Straka, Lukas Miculka, and Zdenek Kotasek. Dependability analysis of fault tolerant systems based on partial dynamic reconfiguration implemented into FPGA. In *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 250–257. IEEE, 2012.

[48] J. P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, pages 243–244. IEEE, 2005.

[49] K.M. Kavi, B.P. Buckles, and U. Narayan Bhat. A formal definition of data flow graph models. *Computers, IEEE Transactions on*, C-35(11):940–948, 1986.

[50] P. Kenterlis, N. Kranitis, A. M. Paschalis, D. Gizopoulos, and M. Psarakis. A low-cost SEU fault emulation platform for SRAM-based FPGAs. In *IOLTS*, pages 235–241, 2006.

[51] V. V. Kumar, R. Verma, J. Lach, and J. Bechta Dugan. A Markov reward model for reliable synchronous dataflow system design. In *Dependable Systems and Networks, 2004 International Conference on*, pages 817–825, 2004.

[52] V.V. Kumar and J. Lach. IC modeling for yield-aware design with variable defect rates. In *Annual Reliability and Maintainability Symposium, 2005. Proceedings*, pages 489–495, 2005.

[53] M. Kwiatkowska, G. Norman, and D. Parker. Controller dependability analysis by probabilistic model checking. *Control Engineering Practice*, 15(11):1427–1434, 2006.

[54] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic model checking for performance and reliability analysis. *SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[55] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: verification of probabilistic real-time systems. In *Computer aided verification*, pages 585–591. Springer, 2011.

[56] Robert Le. Soft error mitigation using prioritized essential bits. *Xilinx XAPP538 (v1. 0)*, 2012.

[57] A. Lesea. Continuing experiments of atmospheric neutron effects on deep submicron integrated circuits (WP286 v1.1), October 2011.

[58] A. Lesea, S. Drimer, J.J. Fabula, C. Carmichael, and P. Alfke. The Rosetta Experiment: Atmospheric Soft Error. Rate Testing in Differing Technology FPGAs. *Device and Materials Reliability, IEEE Transactions on*, 5(3):317–328, Sept 2005.

[59] Daniel Llamocca. Dynamically reconfigurable management of energy, performance, and accuracy applied to digital signal, image, and video processing applications, Ph.D. thesis, The University of New Mexico, USA. 2012.

[60] L. M. Maillart. Maintenance policies for systems with condition monitoring and obvious failures. *IIE Transactions*, 38(6):463–475, 2006.

[61] W. Mansour and R. Velazco. SEU fault-injection in VHDL-based processors: A case study. *J. Electronic Testing*, 29(1):87–94, 2013.

[62] Quinn Martin and Alan D George. Scrubbing optimization via availability prediction (SOAP) for reconfigurable space computing. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, pages 1–6. IEEE, 2012.

[63] Ding Meng, Cao Yun-feng, Wu Qing-xian, and Zhang Zhen. Image processing in optical guidance for autonomous landing of lunar probe. In Agus Budiyono, Bambang Riyanto, and Endra Joelianto, editors, *Intelligent Unmanned Systems: Theory and Applications*, volume 192 of *Studies in Computational Intelligence*, pages 1–10. Springer Berlin Heidelberg, 2009.

[64] John F Meyer. Performability evaluation: Where it is and what lies ahead. In *Computer Performance and Dependability Symposium, 1995. Proceedings., International*, pages 334–343. IEEE, 1995.

[65] Paul S Miner, Victor A Carreño, Mahyar Malekpour, and Wilfredo Torres. A case-study application of RTCA DO-254: design assurance guidance for airborne electronic hardware. In *Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th*, volume 1, pages 1A1–1. IEEE, 2000.

[66] Saraju P Mohanty, Nagarajan Ranganathan, Elias Kougianos, and Priyardarsan Patra. *Low-power high-level synthesis for nanoscale CMOS circuits*. Springer Science & Business Media, 2008.

[67] G.L. Nazar, L.P. Santos, and L. Carro. Scrubbing unit repositioning for fast error repair in FPGAs. In *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*, pages 1–10, Sept 2013.

[68] E. Normand. Single-event effects in avionics. *Nuclear Science, IEEE Transactions on*, 43(2):461–474, Apr 1996.

[69] HA Oldenkamp. Probabilistic model checking: A comparison of tools. 2007.

[70] Takayuki Osogami and Mor Harchol-Balter. Closed form solutions for mapping general distributions to quasi-minimal ph distributions. *Performance Evaluation*, 63(6):524–552, 2006.

[71] Behrooz Parhami. From defects to failures: A view of dependable computing. *ACM SIGARCH Computer Architecture News Special Issue on Architectural Support for Operating Systems*, 16(4):157–168, sep 1988.

[72] SM Parkes. Dsp (demanding space-based processing!): the path behind and the road ahead, dsp98. In *6th International Workshop on Digital Signal Processing Techniques for Space Applications, ESTEC, Noordwijk, The Netherlands*, pages 23–25, 1998.

[73] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 8(6):661–679, 1989.

[74] B. Pratt, M. Caffrey, J.F. Carroll, P. Graham, K. Morgan, and M. Wirthlin. Fine-grain seu mitigation for fpgas using partial tmr. *Nuclear Science, IEEE Transactions on*, 55(4):2274–2280, Aug 2008.

[75] B. H. Pratt, M. P. Caffrey, D. Gibelyou, P. S. Graham, K. Morgan, and M. J. Wirthlin. TMR with More Frequent Voting for Improved FPGA Reliability. In *ERSA*, pages 153–158, 2008.

[76] PRISM. `http://www.prismmodelchecker.org`.

[77] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey. Static proton and heavy ion testing of the Xilinx Virtex-5 device. In *Radiation Effects Data Workshop, 2007 IEEE*, volume 0, pages 177–184, July 2007.

[78] Heather Quinn, Paul S Graham, Keith Morgan, Jim Krone, Michael P Caffrey, and Michael J Wirthlin. An introduction to radiation-induced failure modes and related mitigation methods for Xilinx SRAM FPGAs. In *ERSA*, pages 139–145, 2008.

[79] R. A. Sahner, K. Trivedi, and A. Puliafito. *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Springer Publishing Company, Incorporated, 2012.

[80] P.K. Samudrala, J. Ramos, and S. Katkoori. Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *Nuclear Science, IEEE Transactions on*, 51(5):2957–2969, Oct 2004.

[81] K. Sen, M. Viswanathan, and G. A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *QEST*, volume 5, pages 251–252, 2005.

[82] S Shazli and Mehdi Tahoori. A framework based on boolean satisfiability for soft error rate computation in early design stages. In *Proc. Intl Workshop on Resilience Assessment and Dependability Benchmarking*. Citeseer, 2008.

[83] R. M. Smith, Kishor S. Trivedi, and A. V. Ramesh. Performability analysis: Measures, an algorithm, and a case study. *IEEE Transaction on Computers*, 37(4):406–417, April 1988.

[84] J. D. Snodgrass. Low-power fault tolerance for spacecraft FPGA-based numerical computing, Ph.D. thesis, Naval Postgraduate School, Monterey, California, USA. 2006.

[85] L. Sterpone and A. Ullah. On the optimal reconfiguration times for TMR circuits on SRAM based FPGAs. In *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on*, pages 9–14. IEEE, 2013.

[86] L. Sterpone, M. Violante, R.H. Sorensen, D. Merodio, F. Sturesson, R. Weigand, and S. Mattsson. Experimental Validation of a Tool for Predicting the Effects of Soft Errors in SRAM-Based FPGAs. *Nuclear Science, IEEE Transactions on*, 54(6):2576–2583, Dec 2007.

[87] W. J. Stewart. *Introduction to the numerical solution of Markov Chains*. Princeton University Press, 1994.

[88] G. Swift, C. Carmichael, G. Madias, E. Miller, and R. Monreal. Compendium of XRTC radiation results on all single-event effects observed in the Virtex-5QV. *Proceedings of NASA military and aerospace programmable logic devices (MAPLD)*, pages 1–33, 2011.

[89] L. A. Tambara, F. Almeida, P. Rech, F. L. Kastensmidt, G. Bruni, and C. Frost. Measuring Failure Probability of Coarse and Fine Grain TMR Schemes in SRAM-based FPGAs Under Neutron-Induced Effects. In *Applied Reconfigurable Computing*, volume 9040 of *Lecture Notes in Computer Science*, pages 331–338. Springer International Publishing, 2015.

[90] C. Thibeault, Y. Hariri, S. R. Hasan, C. Hobeika, Y. Savaria, Y. Audet, and F. Z. Tazi. A library-based early soft error sensitivity analysis technique for SRAM-based FPGA design. *J. Electronic Testing*, 29(4):457–471, 2013.

[91] S. Tosun, N. Mansouri, E. Arvas, and Yuan Xie. Reliability-Centric High-Level Synthesis. In *Proc. of DATE*, 2005.

[92] A.J. Tylka, J.H. Adams, P.R. Boberg, B. Brownstein, W.F. Dietrich, E.O. Flueckiger, E.L. Petersen, M.A. Shea, D.F. Smart, and E.C. Smith. CREME96: A revision of the cosmic ray effects on micro-electronics code. *Nuclear Science, IEEE Transactions on*, 44(6):2150–2160, Dec 1997.

[93] Anton FP Van Putten. *Electronic measurement systems: theory and practice.* CRC Press, 1996.

[94] Wim FJ Verhaegh, Paul ER Lippens, Emile HL Aarts, Jan HM Korst, Jef L Van Meerbergen, and Albert van der Werf. Improved force-directed scheduling in high-throughput digital signal processing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(8):945–960, 1995.

[95] Steven Verlinden, Geert Deconinck, and Bernard Coupé. Hybrid reliability model for nuclear reactor safety system. *Reliability Engineering & System Safety*, 101:35–47, 2012.

[96] S.J. Visser, A.S. Dawood, and J.A. Williams. FPGA based real-time adaptive filtering for space applications. In *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on*, pages 322–326, Dec 2002.

[97] Zhong-Ming W., Li-Li D., Zhi-Bin Y., Hong-Xia G., Hui Z., and Min L. The reliability and availability analysis of SEU mitigation techniques in SRAM-based FPGAs. In *Radiation and Its Effects on Components and Systems (RADECS), 2009 European Conference on*, pages 497–503, Sept 2009.

[98] J. W. Warwick, J. B. Pearce, A. C. Riddle, J. K. Alexander, M. D. Desch, M. L. Kaiser, J. R. Thieman, T. D. Carr, S. Gulkis, A. Boischot, et al. Voyager 1 planetary radio astronomy observations near Jupiter. *Science*, 204(4396):995–998, 1979.

[99] D. White. Considerations surrounding single event effects in FPGAs, ASICs, and processors. Technical report, Xilinx Corporation, Tech. Rep, 2011.

[100] Liudong Xing, Suprasad V Amari, and Chaonan Wang. Reliability of k-out-of-n systems with phased-mission requirements and imperfect fault coverage. *Reliability Engineering & System Safety*, 103:45–50, 2012.

[101] Håkan LS Younes. YMER: A statistical model checker. In *Computer Aided Verification*, pages 429–433. Springer, 2005.

Biography

Education

- **Concordia University**: Montreal, Quebec, Canada

  Ph.D candidate, Electrical & Computer Engineering, (Sep. 2011 - date)

- **Concordia University**: Montreal, Quebec, Canada

  M.A.Sc, Electrical & Computer Engineering, (Sep. 2008 - Dec. 2010)

- **Ahsanullah University of Science & Technology**: Dhaka, Bangladesh

  B.Sc, Computer Science & Engineering, (April. 2003 - May. 2007)

Awards

- Concordia Accelerator Award, Concordia University, Canada, 2015

- Travel Grant, The 10th International School on the Effects of Radiation on Embedded Systems for Space Applications (SERESSA), Bariloche, Argentina 2015

- ISU SSP Scholarship, International Space University to attend Space Studies Program (SSP), France, 2014

- FQRNT Doctoral Research Award, Fonds de Recherche du Quebec - Nature et Technologies (FQRNT), Canada, 2012 - 2014
- Deans Funding for Graduate Students, Concordia University, Canada, 2011-2014 and 2008-2010
- Financial Support for Conference Participants, Microsystems Strategic Alliance of Quebec (ReSMiQ), Canada, 2013, 2014 and 2015
- Concordia University Conference and Exposition Award, School of Graduate Studies, Concordia University, Canada, 2013, 2014 and 2015
- ENCS Conference Support, Faculty of Engineering and Computer Science, Concordia University, Canada, 2013
- Best Paper Award, The 8th IEEE International NEWCAS Conference, Canada, 2011
- Student Travel Award, NASA Lunar Science Forum, NASA Ames Research Center, USA, 2010
- ReSMiQ Scholarship for Master Students, Microsystems Strategic Alliance of Quebec (ReSMiQ), Canada, 2009 and 2010

Work History

- **Concordia University**: Montreal, Quebec, Canada

  Research Assistant (Ph.D.), Electrical & Computer Engineering (2011 - date)
- **Bombardier Aerospace**: Montreal, Quebec, Canada

  Research Intern, Core Systems Engineering - Avionics (Summer 2013)

- **Nuance Communications Inc.**: Montreal, Quebec, Canada

  R&D Software Engineering Intern, Embedded Solutions - Automotive (Summer 2011)

- **Concordia University**: Montreal, Quebec, Canada

  Research Assistant (M.A.Sc), Electrical & Computer Engineering (2008 - 2010)

Publications

- ## Journal Papers

  – **Bio-Jr1** K. A. Hoque, O. Ait Mohamed, and Y. Savaria. "Formal Analysis based Early Optimization of Partitioned TMR-Scrubbing for FPGA-based Space Applications using Probabilistic Model Checking", *IEEE Transactions on Systems, Man, and Cybernetics* , Submitted [10 pages].

  – **Bio-Jr2** K. A. Hoque, O. Ait Mohamed, and Y. Savaria. "Formal Analysis of SEU Mitigation for Early Dependability and Performability Analysis of FPGA-based Space Applications", *IEEE Transaction of Dependable and Secure Computing*, Submitted [14 pages].

  – **Bio-Jr3** K. A. Hoque, O. Ait Mohamed, and Y. Savaria. "SAT-MDG: An Automated Safety Checking Methodology", *International Journal of Critical Computer-Based Systems*, volume 3, No.1/2, pages 4 - 25, 2012 [22 pages].

- **Refereed Conference Papers**

  - **Bio-Cf1** K. A. Hoque, O. Ait Mohamed, and Y. Savaria. "A Novel Methodology for High-level Dependability Analysis: Formal Verification Approach", In *10th IEEE Systems Conference (SysCon 2016)*, IEEE, 2016 (accepted).
  - **Bio-Cf2** M. Ammar, K. A. Hoque, and O. Ait Mohamed. "Formal Analysis of Fault Tree using Probabilistic Model Checking: A Solar Array Case Study", In *10th IEEE Systems Conference (SysCon 2016)*, IEEE, 2016 (accepted).
  - **Bio-Cf3** K. A. Hoque, O. Ait Mohamed, and Y. Savaria. "Towards An Accurate Reliability, Availability and Maintainability Analysis Approach for Satellite Systems Based on Probabilistic Model Checking", In *18th ACM/IEEE Design, Automation, and Test in Europe (DATE 2015)*, pages 1635–1640. ACM/IEEE, 2015.
  - **Bio-Cf4** K. A. Hoque, O. Ait Mohamed, Y. Savaria and C. Thibeault. "Probabilistic Model Checking Based DAL Analysis to Optimize a Combined TMR-Blind-Scrubbing Mitigation Technique for FPGA- Based Aerospace Applications", In *12th ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2014)*, pages 175–184. ACM/IEEE, 2015.
  - **Bio-Cf5** K. A. Hoque, O. Ait Mohamed, Y. Savaria and C. Thibeault. "Early Analysis of Soft Error Effects for Aerospace Applications using Probabilistic Model Checking", In *2nd International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2013)*, volume 419 of *CCIS*, pages 54–70. Springer, 2014.

- **Bio-Cf6** K. A. Hoque, O. Ait Mohamed and S. Abed. "SAT Based Model Checking for MDG Models", In *8th IEEE International NEWCAS Conference (NEWCAS 2010)*, pages 241–244. IEEE, 2010.
- **Bio-Cf7** K. A. Hoque, O. Ait Mohamed and S. Abed. "SAT Encoding-Verification Methodology for MDG Models", In *22nd International Conference on Microelectronics (ICM 2010)*, pages 419–422. IEEE, 2010.